NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

FPGA IMPLEMENTATION OF ROBUST SYMMETRICAL NUMBER SYSTEM IN HIGH-SPEED FOLDING ANALOG-TO-DIGITAL CONVERTERS

by

Han Wei Lim

December 2010

Thesis Advisor:	Phillip E. Pace
Second Reader:	David C. Jenn

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 2010	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE FPGA Implementation of Robust Symmetrical Number System in High-Speed Folding Analog-to-Digital Converters	5. FUNDING NUMBERS
6. AUTHOR(S) Han Wei Lim	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Center for Joint Services Electronic Warfare Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) Office of Naval Research, code 31, Washington D.C.	10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number __N.A.__.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT (maximum 200 words)

Analog-To-Digital Converters (ADCs) are integral building blocks of most sensor and communication systems today. As the need for ADCs with faster conversion speeds and lower power dissipation increases, there is a growing motivation to reduce the number of power-consuming components by employing folding circuits to fold the input analog signal symmetrically, prior to quantization by high-speed comparators. These properties of low-power consumption, compactness, high-resolution and fast conversion speeds make folding ADCs an attractive concept to be used for defense applications, such as unmanned systems, direction-finding antenna architectures and system-on-a-chip applications.

In this thesis, a prototype of an optical folding ADC was implemented using the Robust Symmetrical Number System (RSNS). The architecture employs a three-modulus (Moduli 7, 8, 9) scheme to preprocess the antenna signal.

This thesis focuses on the simulation and hardware implementation of this ADC architecture, including the bank of comparators and the RSNS-to-Binary Conversion within a Field Programmable Gate Array (FPGA), to achieve an eight-bit Dynamic Range of 133. This is then integrated with the front-end photonics implementation (designed under a separate thesis).

Low frequency analyses of the results using a 1-kHz input signal indicate a 5.39 Effective Number of Bits (ENOB), a Signal-to-Noise Ratio plus Distortion (SINAD) of 34.21 dB, and a Total Harmonic Distortion (THD) of –61.68 dB.

14. SUBJECT TERMS Analog-to-Digital Converter (ADC), Robust Symmetrical Number System (RSNS), Gray-Code Properties, FPGA	15. NUMBER OF PAGES 129
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

FPGA IMPLEMENTATION OF ROBUST SYMMETRICAL NUMBER SYSTEM IN HIGH-SPEED FOLDING ANALOG-TO-DIGITAL CONVERTERS

Han Wei Lim
Lieutenant Commander, Republic of Singapore Navy
B.E., University of New South Wales, 2002

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
December 2010

Author: Han Wei Lim

Approved by: Phillip E. Pace
Thesis Advisor

David C. Jenn
Second Reader

R. Clark Robertson
Chairman, Department of Electrical and Computer Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Analog-To-Digital Converters (ADCs) are integral building blocks of most sensor and communication systems today. As the need for ADCs with faster conversion speeds and lower power dissipation increases, there is a growing motivation to reduce the number of power-consuming components by employing folding circuits to fold the input analog signal symmetrically prior to quantization by high-speed comparators. These properties of low-power consumption, compactness, high-resolution and fast conversion speeds make folding ADCs an attractive concept to be used for defense applications, such as unmanned systems, direction-finding antenna architectures and system-on-a-chip applications.

In this thesis, a prototype of an optical folding ADC was implemented using the Robust Symmetrical Number System (RSNS). The architecture employs a three-modulus (Moduli 7, 8, 9) scheme to preprocess the antenna signal.

This thesis focuses on the simulation and hardware implementation of this ADC architecture, including the bank of comparators and the RSNS-to-Binary Conversion within a Field Programmable Gate Array (FPGA), to achieve an eight-bit dynamic range of 133. This is then integrated with the front-end photonics implementation (designed under a separate thesis).

Low frequency analyses of the results using a 1-kHz input signal indicate a 5.39 Effective Number of Bits (ENOB), a Signal-to-Noise Ratio plus Distortion (SINAD) of 34.21 dB, and a Total Harmonic Distortion (THD) of -61.68 dB.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Analog-To-Digital Converters (ADCs) are integral building blocks of most sensor and communication systems today. As the need for ADCs with faster conversion speeds and lower power dissipation increases, there is a growing motivation to reduce the number of power-consuming components by employing folding circuits to fold the input analog signal symmetrically prior to quantization by high-speed comparators. The folding of the analog signal allows comparators to be repetitively used, resulting in a smaller die area and lower power consumption.

These properties of low-power consumption, compactness, high-resolution and fast conversion speeds make folding ADCs an attractive concept to be used for defense applications that involve power and size constraints as key factors in the design of battlefield systems and sensors.

In this thesis, a prototype of an optical folding ADC was implemented using the Robust Symmetrical Number System (RSNS), which minimizes the number of comparators and removes the interpolation circuitry completely. The architecture employs a three-modulus (Moduli 7, 8, 9) scheme to preprocess the antenna signal and is shown in Figure 1.

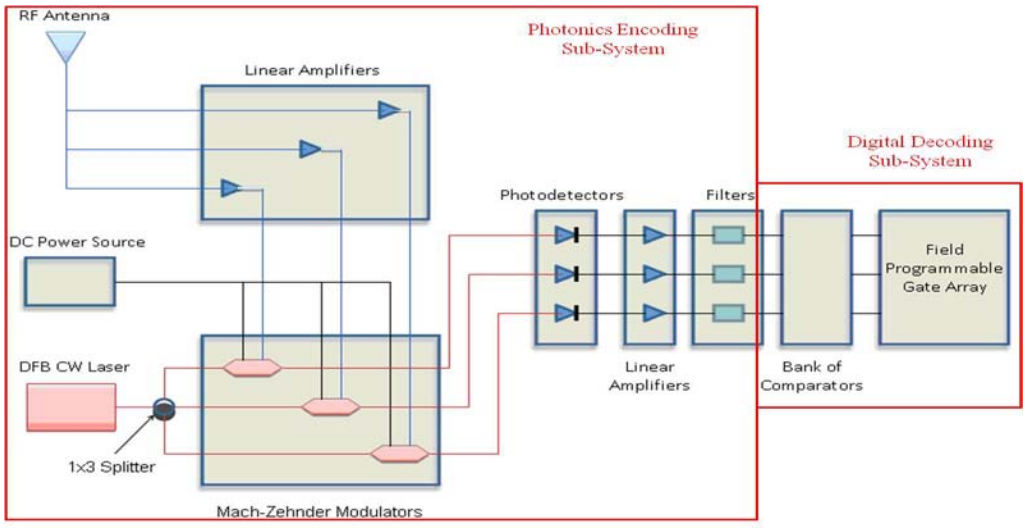

Figure 1. Block Diagram of a Three-Channel Folding ADC Architecture.

The goal of this thesis is to conduct hardware and software implementation of the Digital Decoding Sub-System (DDS) module of the folding ADC architecture (for Moduli 7, 8, 9), from the bank of comparators to the RSNS-to-binary conversion within the Field Programmable Gate Array (FPGA), as well as integration with the front-end Photonics Encoding Sub-System (PES) module of this ADC design. This was accomplished via several milestones described below.

Firstly, the RSNS Dynamic Range (DR) Computation algorithm was verified to be correct, proving that an eight-bit DR of 133 can be achieved theoretically for a three-channel RSNS ADC with Moduli $m_1 = 7$, $m_2 = 8$ and $m_3 = 9$.

Secondly, the RSNS-to-binary conversion algorithm for a three-channel RSNS ADC with Moduli $m_1 = 7$, $m_2 = 8$ and $m_3 = 9$ was developed in LabVIEW, shown in Figure 2.

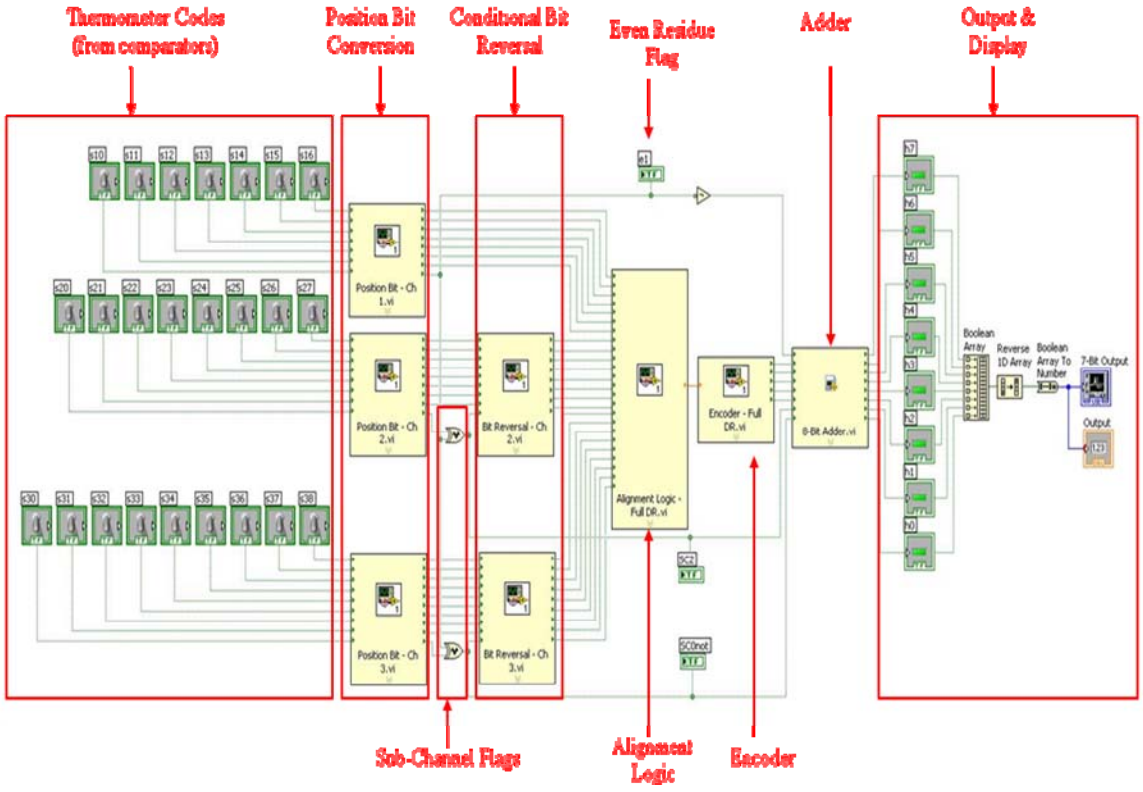

Figure 2. LabVIEW Schematics of RSNS-to-Binary Converter.

This conversion was done to convert the RSNS output into a more convenient decimal representation. This implementation was shown to achieve the DR value of 133 with no ambiguities, which is in agreement with the RSNS DR Computation algorithm, and a one-bit improvement over that achieved in a previous design. Design of thermometer code generator circuits and simulation of this algorithm were carried out to verify that it is working properly before connecting to actual signals.

Thirdly, the comparator circuits and RSNS-to-binary conversion algorithm were designed and implemented on the FPGA to form the DDS module, shown in Figure 3.

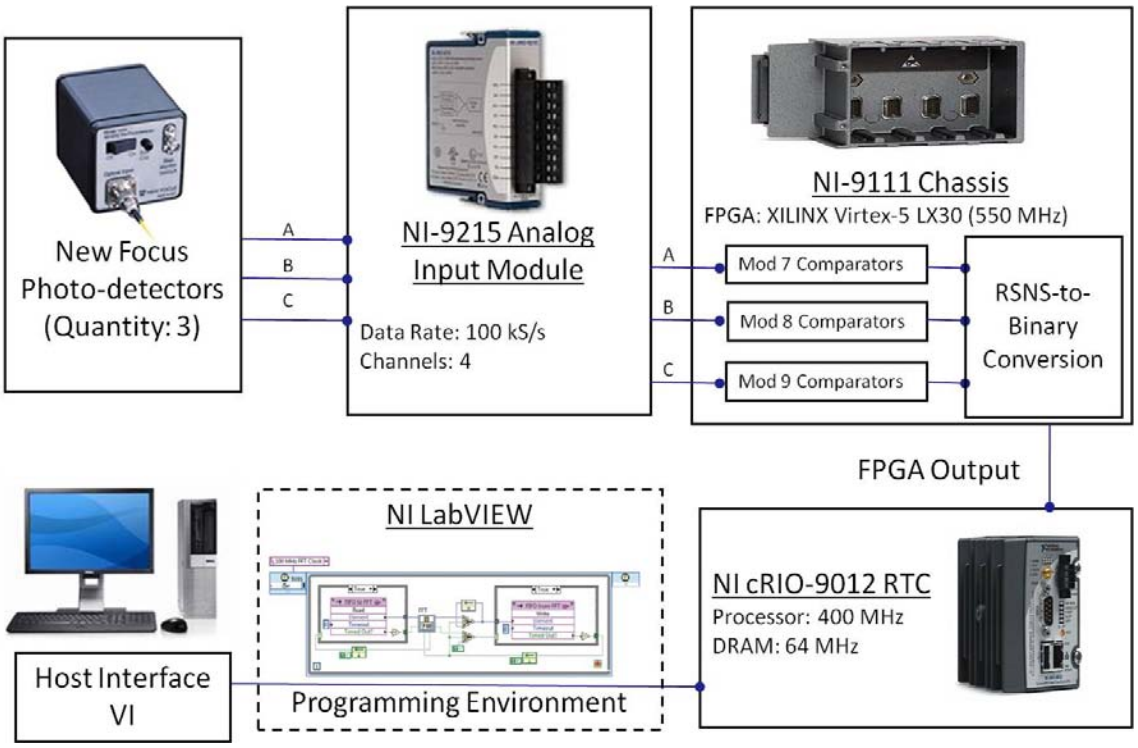

Figure 3. DDS Module Setup

The comparator speed was a limiting factor to the ADC system speed as actual comparator ICs were used for sampling outside the FPGA in a previous design. The key improvement made to the DDS module setup in Figure 3 is that the comparator circuits now reside in the FPGA, allowing them to sample at a rate equal to the FPGA speed.

The comparator circuit and RSNS-to-binary conversion logics were also ran on the FPGA to guarantee a higher FPGA execution speed, as opposed to running it on a National Instruments Real-Time Controller module with a lower processing speed. These two factors allow the ADC to achieve an overall higher sampling frequency.

Lastly, the DDS module was integrated with the front-end PES module to form a folding ADC system and characterization of the ADC performance was carried out. Analysis of the results attributed the dominant noise source in the ADC system to quantization noise, with the ADC remaining resilient to errors caused by other additive noise sources and comparator sampling.

This electro-optic RSNS ADC system has been demonstrated to work and produces an eight-bit output with relatively simple hardware and software. Due to the reduced number of hardware and software components, the energy and size savings make this folding ADC design appealing for defense applications, such as unmanned systems, direction-finding antenna architectures and electronic warfare system-on-a-chip applications.

LIST OF ACRONYMS AND ABBREVIATIONS

ADC	Analog to Digital Converter
ASIC	Application Specific Integrated Circuit
CRIO	Compact Reconfigurable Input/Output
CRT	Chinese Remainder Theorem
DDS	Digital Decoding Sub-System
DMA	Dynamic Memory Allocation
DNL	Differential Non-Linearity
DR	Dynamic Range
ENOB	Effective Number of Bits
FIFO	First-In-First-Out
FPGA	Field Programmable Gate Array
IC	Integrated Circuit
INL	Integral Non-Linearity
I/O	Input/Output
LPS	Least Positive Solution
LSB	Least Significant Bit
NI	National Instruments
PES	Photonics Encoding Sub-System
PRP	Pair-Wise Relatively Prime
RNS	Residue Number System
RSNS	Robust Symmetrical Number System
RTC	Real-Time Controller

SC	Sub-Channel
SINAD	Signal-to-Noise Ratio plus Distortion
SNR	Signal-to-Noise Ratio
THD	Total Harmonics Distortion
VI	Virtual Instrument
$\hat{M}$	Longest series of consecutive non-ambiguous RSNS vectors
P_f	Fundamental period
XR_g	Vector of RNS residues at position g
X_h	Vector of RSNS residues at position h

Table 1. List of Variables.

Variable	**Description**	**Subscript** i	**Subscript** k
g_i	Encoder Output	Bit Order	
h_i	Adder Output	Bit Order	
i_{ik}	RNS States	Channel Number	Bit Order
m_i	Modulus	Channel Number	
n_i	Alignment Logic Output / LPS Solution	Bit Order / Gate Number	
p_{ik}	Position Bit	Channel Number	Bit Order
s_{ik}	Thermometer Code / RSNS States	Channel Number	Bit Order
t_i	Shift Value	Channel Number	

ACKNOWLEDGMENTS

Firstly, I will like to express my gratitude to Singapore's Ministry of Defence and the Republic of Singapore Navy for providing me the opportunity to further my post-graduate education at the Naval Postgraduate School (NPS). NPS is definitely a unique institution to be in, and a melting pot that brings together multiple disciplines and cultures.

It has been my great pleasure and honor to work with Professor Phillip E. Pace and Dr. Brian Luke, who are renowned experts in this field. I will like to express my appreciation for their guidance, tutelage and support during this one-year journey.

I am deeply grateful to Dr James Calusdian, Lab Director of the Photonics Lab, who provided invaluable advice and technical expertise in the realms of FPGA programming. James assisted me in overcoming numerous problems and provided practical suggestions on the next steps towards making the project work. His enthusiasm and resourcefulness provided me with the confidence to complete this project successfully.

I will like to express my most sincere thanks to Kee Leong, my partner-in-crime for this project, and Yean Wee for their numerous brain-storming and helps along the way.

Lastly, I will like to thank my lovely wife, who has graciously endured the lengthy periods when I am away working on this thesis, and for taking good care of me all this while. I will also like to thank my parents and siblings for the support that they have shown me throughout my life, without which I would not be where I am today.

I dedicate this thesis to my first child, wh will be arriving into this world in March 2011.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. FOLDING-TYPE ANALOG-TO-DIGITAL CONVERTERS

Analog-To-Digital Converters (ADCs) are integral building blocks of most sensor and communication systems today. They allow analog data measured in the real world to be sampled and converted into quantized levels for high-speed digital processing.

As the need for ADCs with faster conversion speeds and lower power dissipation increases, there is a growing motivation to reduce the number of power-consuming components by employing folding circuits to fold the input analog signal symmetrically, prior to quantization by high-speed comparators. [1]

The folding of the analog signal allows comparators to be repetitively used, resulting in a smaller die area and lower power consumption. One folding technique employed is the Robust Symmetrical Number System (RSNS), which minimizes the number of comparators and removes the interpolation circuitry completely. [1]

These properties of low-power consumption, compactness, high-resolution and fast conversion speeds make folding ADCs an attractive concept to be used for defense applications, such as unmanned systems, direction-finding antenna architectures and electronic warfare system-on-a-chip applications.

B. SUMMARY OF RECENT RESEARCH

It was demonstrated in [2] that a three-channel folding ADC of Moduli 3, 4 and 5 can be designed using an efficient pipelined RSNS-to-binary algorithm to produce a six-bit Dynamic Range (DR) of 43 while utilizing significantly less electronic components than other equivalent six-bit ADC architecture designs. The implementation of this ADC architecture was verified to have the same DR in [3], shown in Figure 1.

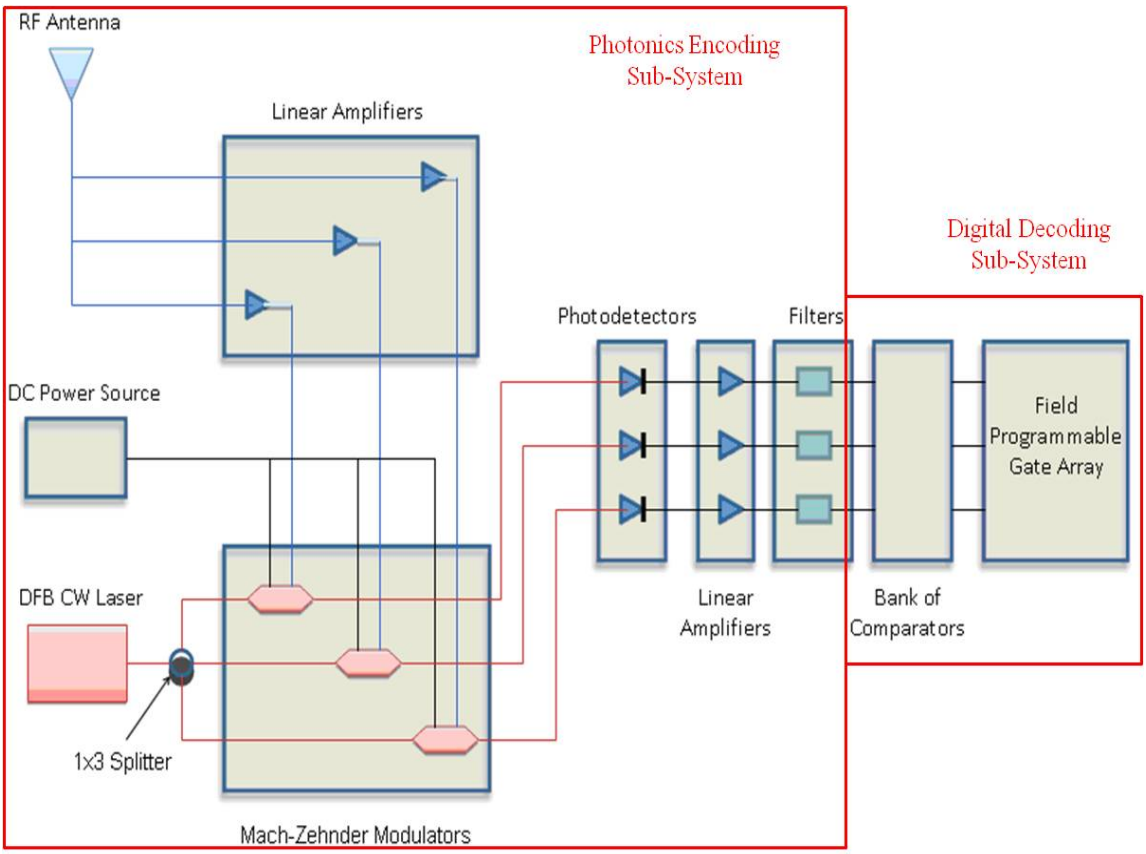

Figure 1. Block Diagram of a Three-Channel Folding ADC Architecture (After [3]).

In [4], it was demonstrated that the same algorithm can be extended and applied to a folding ADC of higher moduli (Moduli 7, 8 and 9). A seven-bit DR of 126 was achieved in [4], but the full DR of 133 could not be achieved due to ambiguities in the ADC output. The next step is to solve for the ambiguities and carry out an implementation of this ADC design.

This thesis is done in conjunction with another Master's thesis to implement a three-channel electro-optical folding ADC architecture (for Moduli 7, 8, 9) [5]. It focuses on the implementation of the Digital Decoding Sub-System (DDS), which involves the hardware and software implementation of the bank of comparators to the RSNS-to-binary conversion within a Field Programmable Gate Array (FPGA), to achieve an eight-bit DR of 133.

This is done using a novel solution to solve for the ambiguities in the ADC output. The DDS module is then integrated with the front-end Photonics Encoding Sub-System (PES), which is detailed in [5].

C. PRINCIPAL CONTRIBUTIONS

Given the works summarized in the previous section, the principal contributions of the research in this thesis are three-fold.

Firstly, this thesis provides a derivation of the theoretical DR and its start and end positions for a folding ADC of Moduli 7, 8 and 9. This is based on the RSNS dynamic range computation algorithm in [6] and [7] and is used as a yardstick comparison to verify with actual experimental values obtained.

The second contribution is the implementation of the RSNS-to-binary algorithm in LabVIEW to achieve an eight-bit DR of 133, which is a one-bit improvement over that achieved in [4]. Simulation of this algorithm is carried out to verify that it is working properly before connecting to actual signals. This is done by designing test circuits to produce the three-channel thermometer codes, which are then supplied to the algorithm to obtain the simulated DR and its position.

Thirdly, this thesis documents the DDS implementation of the comparator circuit design and RSNS-to-binary conversion algorithm onboard a National Instruments (NI) FPGA so as to achieve a higher sampling frequency. The comparator circuit is redesigned in LabVIEW to allow it to achieve a higher speed to match that of the FPGA, as opposed to using actual comparator Integrated Circuits (ICs) as in [3]. The DDS module is then integrated with the front-end PES module in [5] to form an overall folding ADC architecture and tested. Lastly, an analysis and characterization of the ADC performance is carried out.

D. THESIS OUTLINE

Having covered the recent research efforts for a folding ADC architecture, we will provide the reader with a summary of the RSNS structure and principles behind the RSNS-to-binary conversion algorithm in Chapter II. A good understanding of these two

areas is required in order to translate these into actual hardware and software implementation. A theoretical derivation of the DR and its start and end positions for a folding ADC of Moduli 7, 8 and 9, based on the RSNS dynamic range computation algorithm in [6] and [7] is also provided in Chapter II.

All the logic blocks and their equivalent Boolean equations required to form the RSNS-to-binary converter for an ADC of Moduli 7, 8, 9 are delineated in Chapter III.

The process to convert the Boolean equations in Chapter III into a suitable form for implementation in LabVIEW is outlined in Chapter III. It describes how each logic block is designed in LabVIEW and the integration of all logic blocks to form the RSNS-to-binary converter.

The simulation and testing of the RSNS-to-binary converter to verify that the conversion logic is functioning properly is delineated in Chapter V. The first section of this chapter describes the creation of thermometer codes for each channel in LabVIEW to simulate as inputs to the conversion algorithm. The second section highlights the key simulation results obtained for various models.

The process of implementing the comparator circuit design and RSNS-to-binary conversion algorithm on a NI FPGA module is detailed in Chapter VI. It also highlights the integration of the DDS module with the front-end PES module in [5] to form an overall folding ADC architecture are also highlighted.

An analysis of the results to characterize the performance of this ADC system is provided in Chapter VII.

The key conclusions obtained from this project and recommendations for future research are given in Chapter VIII.

II. ROBUST SYMMETRICAL NUMBER SYSTEM

The basic theory and structure of the RSNS, and how its properties are used in the implementation of the RSNS-to-binary converter, is explained in this chapter. Equation Section (Next)

A. RSNS STRUCTURE

A single-channel RSNS is based on the following staircase sequence [1]:

$$X_h = [0,1,2,...m_i - 1, m_i, m_i - 1,...,2,1] \text{ Equation Section (Next)} \quad (1)$$

This sequence starts from zero and increases to a peak value m_i, which is the channel modulus. It then decreases back to zero and repeats itself, forming a periodic sequence with a period of $2m_i$.

A three-channel RSNS vector is denoted as:

$$X_h = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix}, \quad (2)$$

where the RSNS residues are in the range

$$\begin{aligned} s_1 &= \{0, 1, \cdots, m_1\}, \\ s_2 &= \{0, 1, \cdots, m_2\}, \\ s_3 &= \{0, 1, \cdots, m_3\}. \end{aligned} \quad (3)$$

For an N-channel RSNS, each number in the sequence is repeated N times, extending the sequence period to $2Nm_i$. Each RSNS channel modulus is required to be pair-wise relatively prime (PRP) to all other channel moduli [1].

The three-channel RSNS case (N = 3), with channel moduli m_1 = 7, m_2 = 8 and m_3 = 9, which meets the PRP condition, is focused on in this thesis. Each channel period is of length $2Nm_i$. The first 49 RSNS vectors of the three-channel RSNS structure for the moduli $m_i = [7\ 8\ 9]$ are shown in Figure 2 as an illustration.

X_h	$(m_1 = 7)$	0	0	0	1	1	1	2	2	2	3	3	3	4	4	4	5	5	5	6	6	6	7	7	7	6
	$(m_2 = 8)$	0	0	1	1	1	2	2	2	3	3	3	4	4	4	5	5	5	6	6	6	7	7	7	8	8
	$(m_3 = 9)$	0	1	1	1	2	2	2	3	3	3	4	4	4	5	5	5	6	6	6	7	7	7	8	8	8
	h	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24

X_h	$(m_1 = 7)$	6	6	5	5	5	4	4	4	3	3	3	2	2	2	1	1	1	0	0	0	1	1	1	2	…
	$(m_2 = 8)$	8	7	7	7	6	6	6	5	5	5	4	4	4	3	3	3	2	2	2	1	1	1	0	0	…
	$(m_3 = 9)$	9	9	9	8	8	8	7	7	7	6	6	6	5	5	5	4	4	4	3	3	3	2	2	2	…
	h	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	…

Figure 2. Three-Channel RSNS for Moduli $m_i = [7\ 8\ 9]$.

A plot of the folded RSNS waveforms for the three channels in Figure 2, as well as the relative left-shift for Channels 2 and 3, is shown in Figure 3. Notice that these relative shifts are required for the system to exhibit Gray-code properties, where the residues within consecutive RSNS vectors change one at a time at the next code position. This property makes it attractive for error control [1].

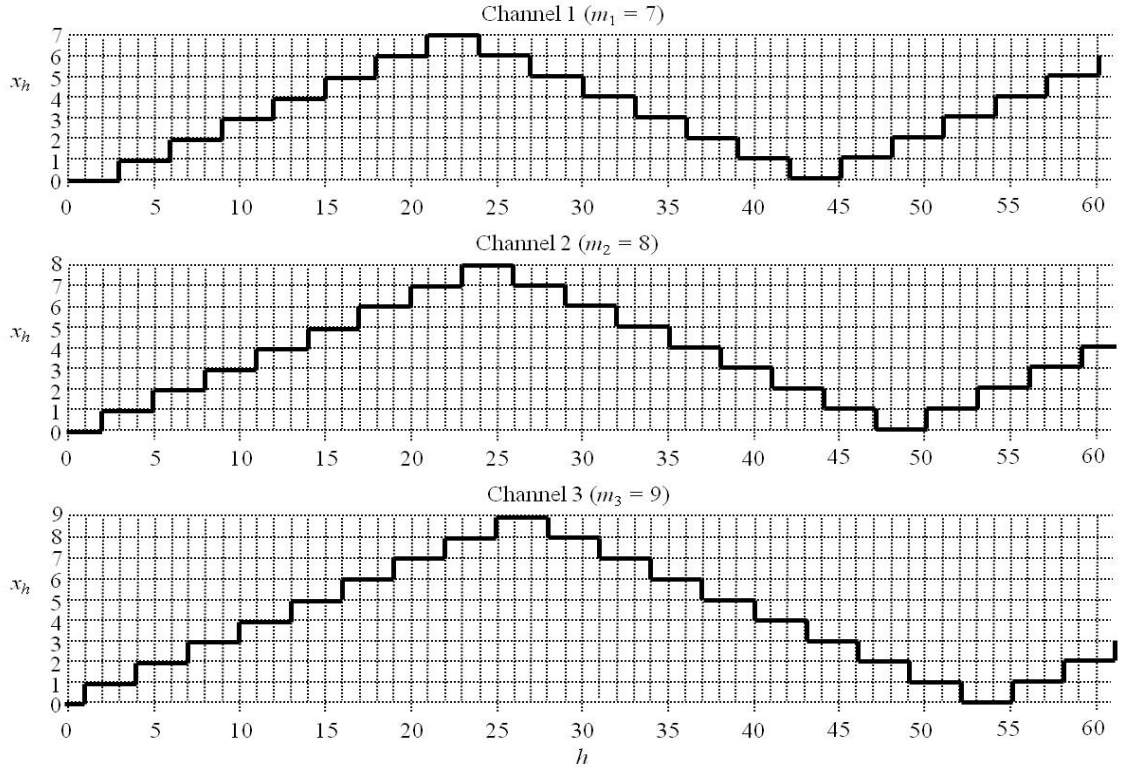

Figure 3. Folded RSNS Waveforms for Moduli $m_i = [7\ 8\ 9]$.

The fundamental period of a three-channel RSNS is [1]:

$$P_f = 2N\prod m_i = 2Nm_1m_2m_3 \tag{4}$$

This is calculated to be $P_f = 2(3)(7)(8)(9) = 3024$ for a RSNS with $m_i = [7\ 8\ 9]$.

1. Dynamic Range

The maximum DR of the RSNS is defined as the longest series of consecutive unambiguous RSNS vectors within the fundamental period of the system [1]. The DR for the same RSNS system is given by [1]:

$$M = \frac{3}{2}m_1^2 + \frac{15}{2}m_1 + 7. \tag{5}$$

This is calculated as $M = \frac{3}{2}(7^2) + \frac{15}{2}(7) + 7 = 133$.

It was shown that the *size* of the DR in a RSNS is the same regardless of the shift sequence and whether the shift is to the left or right. However, the *location* of the DR is affected by the choice of the channels that receive the shifts [1].

Since the aim is to achieve an eight-bit DR for this design regardless of its location, a left-shift $t_i = [0\ 1\ 2]$ system is chosen for ease of implementation as the RSNS-to-binary conversion algorithm is derived based on this shift system in [2], with Channel 2 and 3 shifted one and two positions to the left, respectively, both relative to Channel 1.

2. Ambiguity Types

The conventional approach towards finding the DR and its position is to search the entire fundamental period for a sequence of non-repeating or unambiguous vectors of *unknown* length, which is computationally-intensive. A more efficient approach is to compute the *finite* locations of the ambiguous RSNS vectors for each channel and solve for all vector ambiguity locations to obtain the DR and its position [2], which will be elaborated in Section B of this chapter.

Three types of ambiguities exist in each RSNS channel [2]. The positions of the three ambiguity types for the Channel 1 ($m_1 = 7$) case are illustrated in Figure 4. Type 0 ambiguities occur for every repeating channel period, while Type 1 ambiguities occur on the rise and fall of a channel period. In addition, Type 2 ambiguities occur every time each residue value is repeated three times within the period.

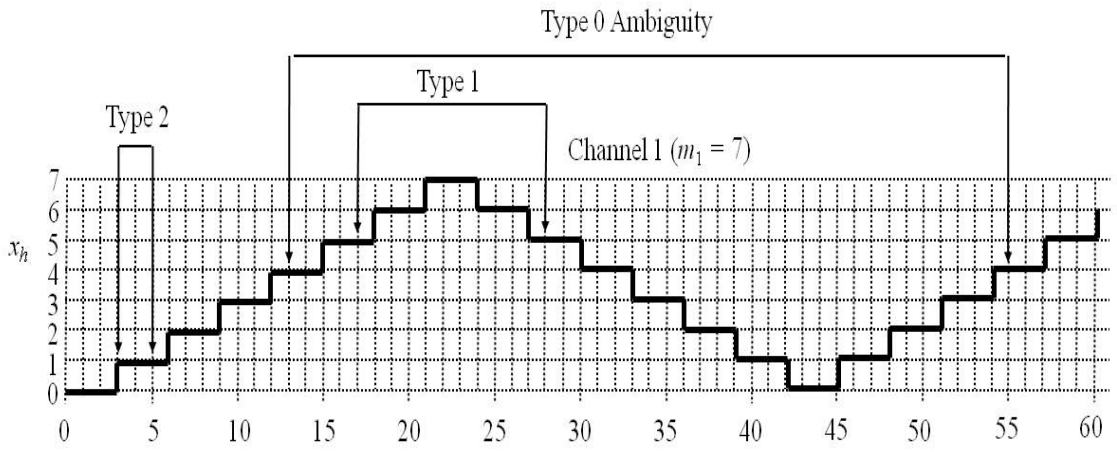

Figure 4. Single Channel Ambiguity Types ($m_1 = 7$).

3. Sub-Channel Analysis

It was shown in [2] that decimating each of the three channels into their sub-channels aids in the solving of channel ambiguities. This is illustrated for the $m_i = [7\ 8\ 9]$ case in Figure 5.

Grouping each of the sub-channels together, and re-indexing the position index h to a new index g, we obtain the results in Figure 6. The relationship between the old position index h and the new index g is given by [2]:

$$\begin{aligned} g &= \frac{h}{3} && \text{(Sub-Channel 0)} \\ g &= \frac{(h-1)}{3} && \text{(Sub-Channel 1)} \\ g &= \frac{(h-2)}{3} && \text{(Sub-Channel 2)} \end{aligned} \tag{6}$$

m_1=7	X_h	0	0	0	1	1	1	2	2	2	3	3	3	4	4	4	5	5	5
Sub-Channel 0		0			1			2			3			4			5		
Sub-Channel 1			0			1			2			3			4			5	
Sub-Channel 2				0			1			2			3			4			5
Position Index	*h*	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

m_2=8	X_h	0	0	1	1	1	2	2	2	3	3	3	4	4	4	5	5	5	6
Sub-Channel 0		0			1			2			3			4			5		
Sub-Channel 1			0			1			2			3			4			5	
Sub-Channel 2				1			2			3			4			5			6
Position Index	*h*	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

m_3=9	X_h	0	1	1	1	2	2	2	3	3	3	4	4	4	5	5	5	6	6
Sub-Channel 0		0			1			2			3			4			5		
Sub-Channel 1			1			2			3			4			5			6	
Sub-Channel 2				1			2			3			4			5			6
Position Index	*h*	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Figure 5. Decimation of Channels into Sub-Channels for Moduli $m_i = [7\ 8\ 9]$.

Sub-Channel 0

m_1=7		0	1	2	3	4	5	6	7	6	5	4	3	2	1	0	1	2	3
m_2=8	X_h	0	1	2	3	4	5	6	7	8	7	6	5	4	3	2	1	0	1
m_3=9		0	1	2	3	4	5	6	7	8	9	8	7	6	5	4	3	2	1
Position Index	*h*	0	3	6	9	12	15	18	21	24	27	30	33	36	39	42	45	48	51
	g = *h*/3	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Sub-Channel 1

m_1=7		0	1	2	3	4	5	6	7	6	5	4	3	2	1	0	1	2	3
m_2=8	X_h	0	1	2	3	4	5	6	7	8	7	6	5	4	3	2	1	0	1
m_3=9		1	2	3	4	5	6	7	8	9	8	7	6	5	4	3	2	1	0
Position Index	*h*	1	4	7	10	13	16	19	22	25	28	31	34	37	40	43	46	49	52
	g = (*h*-1)/3	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Sub-Channel 2

m_1=7		0	1	2	3	4	5	6	7	6	5	4	3	2	1	0	1	2	3
m_2=8	X_h	1	2	3	4	5	6	7	8	7	6	5	4	3	2	1	0	1	2
m_3=9		1	2	3	4	5	6	7	8	9	8	7	6	5	4	3	2	1	0
Position Index	*h*	2	5	8	11	14	17	20	23	26	29	32	35	38	41	44	47	50	53
	g = (*h*-2)/3	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Figure 6. RSNS Vectors for Sub-Channels 0, 1 and 2.

A plot of the RSNS sub-channel waveforms in Figure 6 is shown in Figure 7. Each channel has $N = 3$ sub-channels. From Figure 7, it can be seen that Type 2 ambiguities are eliminated, leaving only Type 0 and 1 ambiguities.

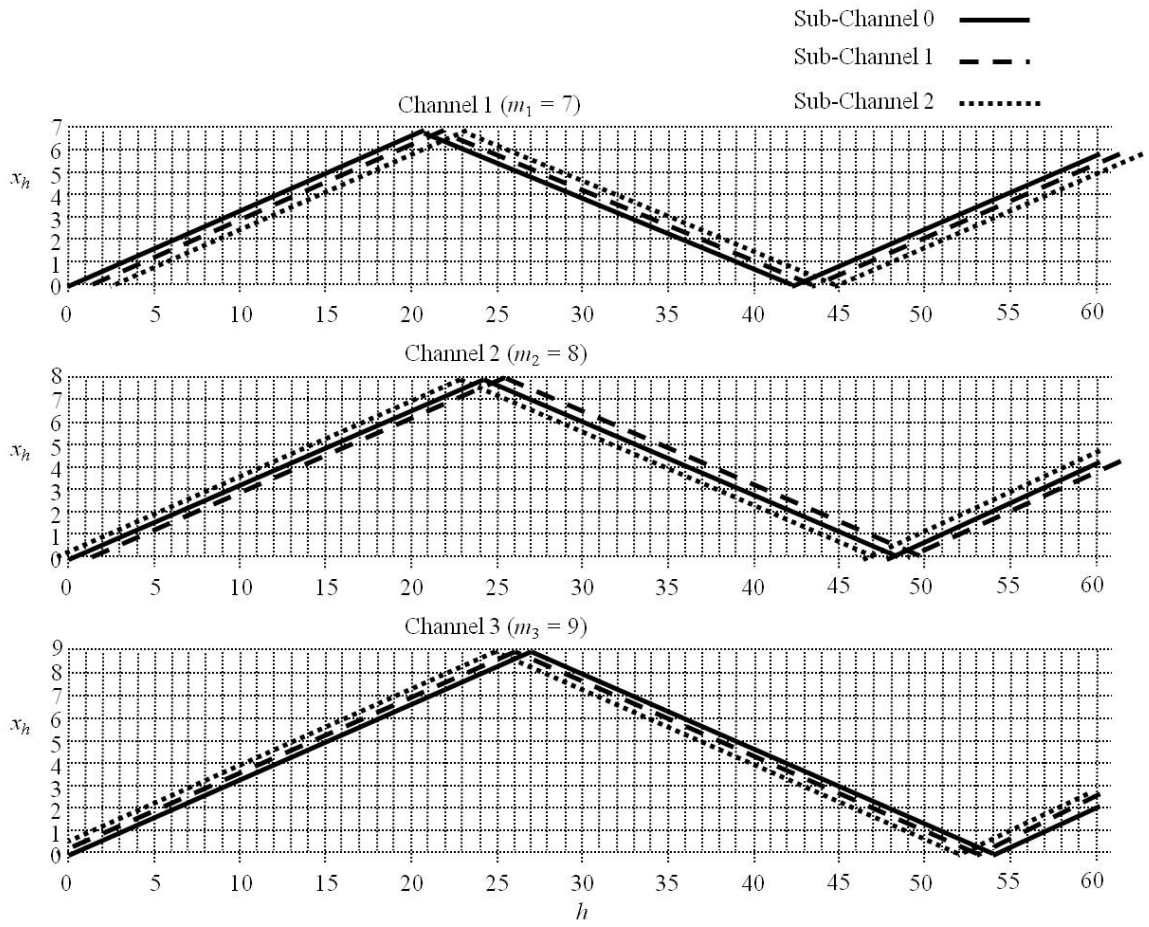

Figure 7. Plot of Sub-Channels 0, 1 and 2.

As the waveforms are folded and symmetric, each residue value occurs twice in a single folding period, except for the minimum and the maximum value, which occur only once. Using this fact, we obtain the congruence equations describing the position of a RSNS residue vector for Sub-Channel 0 as [2]:

$$\begin{aligned} g &\equiv s_1 \pmod{2m_1} \quad \text{or} \quad g \equiv 2m_1 - s_1 \pmod{2m_1}, \\ g &\equiv s_2 \pmod{2m_2} \quad \text{or} \quad g \equiv 2m_2 - s_2 \pmod{2m_2}, \\ g &\equiv s_3 \pmod{2m_3} \quad \text{or} \quad g \equiv 2m_3 - s_3 \pmod{2m_3}. \end{aligned} \tag{7}$$

The congruence equations for Sub-Channel 1 are:

$$\begin{aligned} g &\equiv s_1 \pmod{2m_1} \quad \text{or} \quad g \equiv 2m_1 - s_1 \pmod{2m_1}, \\ g &\equiv s_2 \pmod{2m_2} \quad \text{or} \quad g \equiv 2m_2 - s_2 \pmod{2m_2}, \\ g &\equiv s_3 - 1 \pmod{2m_3} \quad \text{or} \quad g \equiv 2m_3 - s_3 - 1 \pmod{2m_3}. \end{aligned} \tag{8}$$

The congruence equations for Sub-Channel 2 are:

$$\begin{aligned} g &\equiv s_1 \pmod{2m_1} \quad \text{or} \quad g \equiv 2m_1 - s_1 \pmod{2m_1}, \\ g &\equiv s_2 - 1 \pmod{2m_2} \quad \text{or} \quad g \equiv 2m_2 - s_2 - 1 \pmod{2m_2}, \\ g &\equiv s_3 - 1 \pmod{2m_3} \quad \text{or} \quad g \equiv 2m_3 - s_3 - 1 \pmod{2m_3}. \end{aligned} \tag{9}$$

Equations (7)–(9) show that there are three equations for each sub-channel, with two choices for each equation. Thus, each RSNS residue vector can produce up to 2^3 or eight unique systems of equations. This means that any RSNS vector can have up to eight ambiguities within the fundamental period.

The equations also reveal that the start position of the DR, *h*, can be found by determining the sub-channel of a particular RSNS vector, solving the equivalent set of equations to obtain *g*, and converting *g* to find *h*, using (6). To determine which sub-channel a RSNS vector is from, the even-odd structure of each sub-channel has to be investigated.

4. Even-Odd Analysis

The even-odd structures (*e* = even, *o* = odd) of each sub-channel, and the overall even-odd structure of a three-channel RSNS with $m_i = [7\ 8\ 9]$, are illustrated in Figures 8 and 9.

From Figure 8, it can be seen that each sub-channel produces RSNS vectors with two unique even-odd structures. Thus, the Sub-Channel 0 equations in (7) can be applied if the RSNS vector is found to be of form $[e\ e\ e]^T$ or $[o\ o\ o]^T$. Similarly, the Sub-Channel 1 equations in (8) are used if the RSNS vector is of form $[e\ e\ o]^T$ or $[o\ o\ e]^T$, and the Sub-Channel 2 equations in (9) are applied if the RSNS vector is of form $[e\ o\ o]^T$ or $[o\ e\ e]^T$.

Sub-Channel 0

m_1=7	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*
m_2=8	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*
m_3=9	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*
h	0	3	6	9	12	15	18	21	24	27	30	33	36	39	42	45	48	51
g = h/3	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Sub-Channel 1

m_1=7	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*
m_2=8	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*
m_3=9	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*
h	1	4	7	10	13	16	19	22	25	28	31	34	37	40	43	46	49	52
g = (*h*-1)/3	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Sub-Channel 2

m_1=7	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*
m_2=8	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*
m_3=9	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*	*o*	*e*
h	2	5	8	11	14	17	20	23	26	29	32	35	38	41	44	47	50	53
g = (*h*-2)/3	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Figure 8. Even-Odd Structure of RSNS Vectors for Sub-Channels 0, 1 and 2.

This even-odd structure repeats in blocks of six, after cycling through the set of sub-channels twice, as shown in Figure 9. This means that the minimum distance between ambiguous parity vectors is always a multiple of six for a three-channel RSNS, and a multiple of 2*N* for the *N*-channel case.

Sub-Channel	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2	0	1	2
m_1=7	*e*	*e*	*e*	*o*	*o*	*o*	*e*	*e*	*e*	*o*	*o*	*o*	*e*	*e*	*e*	*o*	*o*	*o*
m_2=8	*e*	*e*	*o*	*o*	*o*	*e*	*e*	*e*	*o*	*o*	*o*	*e*	*e*	*e*	*o*	*o*	*o*	*e*
m_3=9	*e*	*o*	*o*	*o*	*e*	*e*	*e*	*o*	*o*	*o*	*e*	*e*	*e*	*o*	*o*	*o*	*e*	*e*
h	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Figure 9. Overall Even-Odd Structure of RSNS Vectors.

It was shown in [2] that for the Sub-Channel 1 and 2 cases, the equations in (6) and (7) actually convert the even-odd structure of the RSNS vectors to the form $\left[e\ e\ e\right]^T$ or $\left[o\ o\ o\right]^T$. Thus, the Sub-Channel 0 case can be exploited as a base case to develop an efficient RSNS-binary conversion, explained in Section C of this chapter.

B. RSNS DYNAMIC RANGE SEARCH ALGORITHM

An efficient RSNS dynamic range search algorithm was developed in [6] and [7], which makes use of the cyclical and symmetrical properties of the RSNS structure to cut down on the solution space when searching for the DR. This is faster than searching through the whole solution space, which is computationally intensive. A theoretical derivation of the DR, including its start and end positions, for a folding ADC of Moduli 7, 8 and 9, based on this algorithm, is provided in this section.

1. Dynamic Range Upper Bound

The upper limit of $\hat{M}$ is defined as [6]:

$$\left\lceil \hat{M} \right\rceil = N \min_{\forall i \forall j} \left[\prod_{\forall i} m_i + 2 \prod_{\forall j} m_j \right] - 1 \tag{10}$$

The DR upper bound for an ADC of Moduli 7, 8 and 9 is calculated to be 221:

$$\begin{aligned}
\left\lceil \hat{M} \right\rceil_1 &= 3\left[1 + 2\left(7 \cdot 8 \cdot 9\right)\right] - 1 = 3026 \\
\left\lceil \hat{M} \right\rceil_2 &= 3\left[7 + 2\left(8 \cdot 9\right)\right] - 1 = 452 \\
\left\lceil \hat{M} \right\rceil_3 &= 3\left[8 + 2\left(7 \cdot 9\right)\right] - 1 = 401 \\
\left\lceil \hat{M} \right\rceil_4 &= 3\left[9 + 2\left(7 \cdot 8\right)\right] - 1 = 362 \\
\left\lceil \hat{M} \right\rceil_5 &= 3\left[\left(7 \cdot 8\right) + 2\left(9\right)\right] - 1 = 221 \\
\left\lceil \hat{M} \right\rceil_6 &= 3\left[\left(7 \cdot 9\right) + 2\left(8\right)\right] - 1 = 236 \\
\left\lceil \hat{M} \right\rceil_7 &= 3\left[\left(8 \cdot 9\right) + 2\left(7\right)\right] - 1 = 257 \\
\left\lceil \hat{M} \right\rceil_8 &= 3\left[\left(7 \cdot 8 \cdot 9\right) + 2\left(1\right)\right] - 1 = 1517 \\
\therefore \left\lceil \hat{M} \right\rceil &= \min \left\lceil \hat{M} \right\rceil_i = 221
\end{aligned} \tag{11}$$

2. RSNS Vector Ambiguity Locations

The solutions to the RSNS vector ambiguity locations for a three-channel case are shown in Figure 10. Note that all of the ambiguities smaller than the fundamental period are symmetric around a Center of Ambiguity (COA), as intuitively shown in Figure 4 [6].

The three identifier digits in the Case Label refer to the number of Type 1 ambiguities, combination number and sub-channel number respectively. They are used to provide a logical case numbering system for implementation in a computer algorithm [2].

Case Label	Ambiguities occur at h and $h+k$, where h is	k is a multiple of	COR
010	Any position in the fundamental period	$6m_1m_2m_3$	N/A
110	$h = a(3m_1) - k/2$	$6m_2m_3$	$a(3m_1)$
111	$h = a(3m_1) + 1 - k/2$		$a(3m_1) + 1$
112	$h = a(3m_1) + 2 - k/2$		$a(3m_1) + 2$
120	$h = a(3m_2) - k/2$	$6m_1m_3$	$a(3m_2)$
121	$h = a(3m_2) + 1 - k/2$		$a(3m_2) + 1$
122	$h = a(3m_2) - 1 - k/2$		$a(3m_2) - 1$
130	$h = a(3m_3) - k/2$	$6m_1m_2$	$a(3m_3)$
131	$h = a(3m_3) - 2 - k/2$		$a(3m_3) - 2$
132	$h = a(3m_3) - 1 - k/2$		$a(3m_3) - 1$
210	$h = a(3m_1m_2) - k/2$	$6m_3$	$a(3m_1m_2)$
211	$h = a(3m_1m_2) + h_{s1} - k/2$		$a(3m_1m_2) + h_{s1}$
212	$h = a(3m_1m_2) + h_{s2} - k/2$		$a(3m_1m_2) + h_{s2}$
220	$h = a(3m_1m_3) - k/2$	$6m_2$	$a(3m_1m_3)$
221	$h = a(3m_1m_3) + h_{s1} - k/2$		$a(3m_1m_3) + h_{s1}$
222	$h = a(3m_1m_3) + h_{s2} - k/2$		$a(3m_1m_3) + h_{s2}$
230	$h = a(3m_2m_3) - k/2$	$6m_1$	$a(3m_2m_3)$
231	$h = a(3m_2m_3) + h_{s1} - k/2$		$a(3m_2m_3) + h_{s1}$
232	$h = a(3m_2m_3) + h_{s2} - k/2$		$a(3m_2m_3) + h_{s2}$
310	$h = a(3m_1m_2m_3) - k/2$	6	$a(3m_1m_2m_3)$
311	$h = a(3m_1m_2m_3) + h_{s1} - k/2$		$a(3m_1m_2m_3) + h_{s1}$
312	$h = a(3m_1m_2m_3) + h_{s2} - k/2$		$a(3m_1m_2m_3) + h_{s2}$

Figure 10. RSNS Vector Ambiguity Locations (Three-Channel Case).

From Figure 10, it can be seen that the ambiguities in an RSNS structure occur at position h and $h + k$, where k is a multiple of the moduli combinations for each case. There is an inverse relation between the spacing between the ambiguous vectors (k) and the spacing between the COA.

Recall that a left-shift $shift_i = [0\ 1\ 2]$ system was implemented to exhibit gray-code properties. Thus, the base cases involving Sub-Channel 0 (Case XX0) has a COA shift of $h_{s0} = 0$. The cases involving Sub-Channel 1 (Case XX1) and Sub-Channel 2 (Case XX2) will have a COA shifts of h_{s1} and h_{s2}, respectively. These are the least positive solutions to the following two sets of congruence equations [7]:

$$\begin{array}{ll} \frac{h_{s1}-1}{3} \equiv 0 \pmod 7 & \frac{h_{s2}-2}{3} \equiv 0 \pmod 7 \\ \frac{h_{s1}-1}{3} \equiv 0 \pmod 8 & \frac{h_{s2}+1}{3} \equiv 0 \pmod 8 \\ \frac{h_{s1}+2}{3} \equiv 0 \pmod 9 & \frac{h_{s2}+1}{3} \equiv 0 \pmod 9 \end{array} \qquad (12)$$

The COA shifts are computed to be $h_{s1} = 1$ and $h_{s2} = 23$ using the generalized Chinese Remainder Theorem (CRT) procedure described in the Appendix.

The solutions to the RSNS vector ambiguity locations for a RSNS with Moduli 7, 8 and 9, are derived from Figure 10 and summarized in Figure 11. The rows highlighted in grey have ambiguity pairs with a length greater than $\left\lceil \frac{M}{2} \right\rceil = 221$ and can be ignored in the DR computation.

Case Label	Ambiguities occur at h and $h+k$, where h is	k is a multiple of	COA
010	Any position in the fundamental period	3024	N/A
110	$h = a(21) - 216$	432	$a(21)$
111	$h = a(21) - 215$		$a(21) + 1$
112	$h = a(21) - 214$		$a(21) + 2$
120	$h = a(24) - 189$	378	$a(24)$
121	$h = a(24) - 188$		$a(24) + 1$
122	$h = a(24) - 190$		$a(24) - 1$
130	$h = a(27) - 168$	336	$a(27)$
131	$h = a(27) - 170$		$a(27) - 2$
132	$h = a(27) - 169$		$a(27) - 1$
210	$h = a(168) - 27$	54	$a(168)$
211	$h = a(168) + h_{s1} - 27$		$a(168) + h_{s1}$
212	$h = a(168) + h_{s2} - 27$		$a(168) + h_{s2}$
220	$h = a(189) - 24$	48	$a(189)$
221	$h = a(189) + h_{s1} - 24$		$a(189) + h_{s1}$
222	$h = a(189) + h_{s2} - 24$		$a(189) + h_{s2}$
230	$h = a(216) - 21$	42	$a(216)$
231	$h = a(216) + h_{s1} - 21$		$a(216) + h_{s1}$
232	$h = a(216) + h_{s2} - 21$		$a(216) + h_{s2}$
310	$h = a(1512) - 3$	6	$a(1512)$
311	$h = a(1512) + h_{s1} - 3$		$a(1512) + h_{s1}$
312	$h = a(1512) + h_{s2} - 3$		$a(1512) + h_{s2}$

Figure 11. RSNS Vector Ambiguity Locations (for Moduli 7, 8 and 9).

3. Minimal Ambiguity Pair Locations

Due to the symmetry of the RSNS vector ambiguity locations about $P_f / 2$, only ambiguity pairs from $h = -N$ to $\left(P_f / 2\right) + N$ need to be considered when computing M [7]. All the minimal ambiguity pair (h_1, h_2) locations, derived by substituting integer values of a, $h_{s1} = 1$ and $h_{s2} = 23$ into Figure 11 for each of the cases, are shown in Figure 12, for $h = -3$ to 1515.

h_1	COA	h_2	Case
-3	0	3	310
-26	1	28	211
-4	23	50	212
4	25	46	231
82	106	130	221
83	107	131	222
141	168	195	210
142	169	196	211
165	189	213	220
194	215	236	232
195	216	237	230
220	241	262	231
271	295	319	221
272	296	320	222
309	336	363	210
310	337	364	211
332	359	386	212
354	378	402	220
410	431	452	232
411	432	453	230
436	457	478	231
460	484	508	221
461	485	509	222
477	504	531	210
478	505	532	211
500	527	554	212
543	567	591	220
626	647	668	232
627	648	669	230

h_1	COA	h_2	Case
670	673	676	311
732	756	780	220
860	863	866	312
868	889	910	231
921	945	969	220
981	1008	1035	210
982	1009	1036	211
1004	1031	1058	212
1027	1051	1075	221
1028	1052	1076	222
1058	1079	1100	232
1059	1080	1101	230
1084	1105	1126	231
1110	1134	1158	220
1149	1176	1203	210
1150	1177	1204	211
1172	1199	1226	212
1216	1240	1264	221
1217	1241	1265	222
1274	1295	1316	232
1275	1296	1317	230
1300	1321	1342	231
1317	1344	1371	210
1318	1345	1372	211
1340	1367	1394	212
1405	1429	1453	221
1406	1430	1454	222
1509	1512	1515	310

Figure 12. Minimal Pair Locations.

4. Consecutive Minimal Pair Locations

In Figure 12, all minimal pairs with a starting position h_1 earlier than the starting position of the previous pair are removed. The remaining minimal pairs are sorted such that h_2 is monotonically increasing and are defined as *consecutive* minimal pairs [7]. These consecutive minimal pairs are derived from Figure 12 and shown in Figure 13.

h_1	COA	h_2	Case	Distance Between Consecutive Minimal Pairs
-3	0	3	310	N.A.
4	25	46	231	48
82	106	130	221	125
83	107	131	222	48
141	168	195	210	111
142	169	196	211	54
165	189	213	220	70
194	215	236	232	70
195	216	237	230	42
220	241	262	231	66
271	295	319	221	98
272	296	320	222	48
309	336	363	210	90
310	337	364	211	54
332	359	386	212	75
354	378	402	220	69
410	431	452	232	97
411	432	453	230	42
436	457	478	231	66
460	484	508	221	71
461	485	509	222	48
477	504	531	210	69
478	505	532	211	54
500	527	554	212	75
543	567	591	220	90
626	647	668	232	124
627	648	669	230	42
670	673	676	311	48
732	756	780	220	109
860	863	866	312	133
868	889	910	231	49
921	945	969	220	100
981	1008	1035	210	113
982	1009	1036	211	54
1004	1031	1058	212	75
1027	1051	1075	221	70
1028	1052	1076	222	48
1058	1079	1100	232	71
1059	1080	1101	230	42
1084	1105	1126	231	66
1110	1134	1158	220	73
1149	1176	1203	210	92
1150	1177	1204	211	54
1172	1199	1226	212	75
1216	1240	1264	221	91
1217	1241	1265	222	48
1274	1295	1316	232	98
1275	1296	1317	230	42
1300	1321	1342	231	66
1317	1344	1371	210	70
1318	1345	1372	211	54
1340	1367	1394	212	75
1405	1429	1453	221	112
1406	1430	1454	222	48
1509	1512	1515	310	108

Figure 13. Consecutive Minimal Pair Locations.

The DR M is the largest distance between endpoints of two consecutive minimal pairs, computed in Figure 13. The result is a $M = 133$ starting at $h = h_1 + 1 = 733$ and ending at $h = h_2 - 1 = 865$, which is in agreement with (5).

C. RSNS-TO-BINARY CONVERSION ALGORITHM

With a good understanding of the underlying RSNS structure, an efficient RSNS-to-binary conversion can be achieved by exploiting the relationship between the RSNS and the Residue Number System (RNS) and using the RNS Least Positive Solution (LPS) and positional alignment techniques to solve for the DR position. These are summarized in this section.

1. RSNS-RNS Relationship

It was shown in [2] that there is a one-to-one correspondence between the RSNS and the RNS. This is the key to achieving an efficient RSNS-to-binary conversion as the RNS has no ambiguities in a fundamental period, unlike the RSNS.

This one-to-one correspondence between the RSNS and RNS vectors, using the Sub-Channel 0 case as a base case for conversion, is demonstrated in Figure 14. Each RSNS residue is converted to a unique RNS residue such that there is no ambiguity within a single channel period.

Sub-Channel 0

RSNS	m_1=7	0	1	2	3	4	5	6	7	6	5	4	3	2	1	0	1	2	3
Vectors	m_2=8	0	1	2	3	4	5	6	7	8	7	6	5	4	3	2	1	0	1
X_g	m_3=9	0	1	2	3	4	5	6	7	8	9	8	7	6	5	4	3	2	1
RNS	m_1=7	0	1	2	3	4	5	6	7	8	9	10	11	12	13	0	1	2	3
Vectors	m_2=8	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0	1
XR_g	m_3=9	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
	g	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Figure 14. One-to-One Correspondence between RSNS and RNS Vectors.

Another useful RNS property is that every XR_g vector is either all even or odd, unlike the RSNS.

It is possible to simplify the conversion by considering only the even XR_g vectors (grey-colored columns in Figure 14) and dividing their values and the index g by two. This is illustrated in Figure 15.

RSNS	m_1=7	0	2	4	6	6	4	2	0	2
Vectors	m_2=8	0	2	4	6	8	6	4	2	0
$X_{g/2}$	m_3=9	0	2	4	6	8	8	6	4	2
	g	0	2	4	6	8	10	12	14	16
RNS	m_1=7	0	1	2	3	4	5	6	0	1
Vectors	m_2=8	0	1	2	3	4	5	6	7	0
$XR_{g/2}/2$	m_3=9	0	1	2	3	4	5	6	7	8
	$g/2$	0	1	2	3	4	5	6	7	8

Figure 15. RSNS and RNS Vectors for Even Index g.

From Figure 15, it can be seen that the RSNS vectors are transformed into RNS vectors with the same PRP moduli. The index position $g/2$ can now be solved directly using the standardized CRT [10]. However, there are still up to 2^3 or eight systems of equations to solve in order to find the positions of a single RSNS vector within the fundamental period. This is due to the symmetry of each RSNS channel period and three sub-channel structure within each channel period.

This process can be simplified by limiting the solution range to within the DR. The problem is then reduced to finding the least positive solution of the eight systems of equations, explained in the next section.

2. Dynamic Range Compression

A closed-form solution for the start and end DR positions for moduli of form (2^r – 1, 2^r, 2^r + 1) was derived in [6]. The radix r is 3 for an RSNS with moduli $m_i = [7\ 8\ 9]$. The start and end positions can be calculated using [6]:

$$\text{Start Position of } \hat{M}\text{: } h_1 = 3\left(2^{3r-1} - 2^{r-1} - 2^r\right) + 1 \tag{13}$$

and

$$\text{End Position of } \hat{M}\text{: } h_2 = 3\left(2^{3r-1} + 2^{2r-1}\right) + 1. \tag{14}$$

The DR range is from $h_1 = 733$ and $h_2 = 865$ for a RSNS with $m_i = \begin{bmatrix} 7 & 8 & 9 \end{bmatrix}$. The corresponding RSNS vectors are $X_{733} = [6\ 4\ 7]^T$ and $X_{865} = [6\ 0\ 1]^T$ using the method described in [5].

Recall that this algorithm is applied to the $\begin{bmatrix} e\ e\ e \end{bmatrix}^T$ vectors in the Sub-Channel 0 case only. Thus, the closest $\begin{bmatrix} e\ e\ e \end{bmatrix}^T$ vectors to X_{733} and X_{865} must be chosen for this conversion to work. They are $X_{732} = [6\ 4\ 8]^T$ and $X_{864} = [6\ 0\ 0]^T$.

At this juncture, it must be noted that the chosen start position $h_1 = 732$ falls outside the DR, which will produce ambiguous results during implementation. This is a special case where the DR does not start with a Sub-Channel 0 vector and end with a Sub-Channel 2 vector, as required in Figure 9.

There are two methods to overcome this. The first is to truncate the DR to a multiple of six, so that the DR will always start with a Sub-Channel 0 vector. Alternatively, the boundary (start and end) vectors have to be treated as special cases in the LabVIEW implementation. The details of both methods will be addressed in Chapters IV and V.

Applying (6), we calculate the corresponding start and end indices for the RNS XR_g vectors as:

$$g_1 = \frac{h_1}{3} = \frac{732}{3} = 244, \qquad g_2 = \frac{h_2}{3} = \frac{864}{3} = 288. \tag{15}$$

Using the results from Figure 14, we consider only the even RNS XR_g vectors. The corresponding start and end indices for the RNS $XR_{g/2} / 2$ vectors are calculated as:

$$\frac{g_1}{2} = \frac{244}{2} = 122, \qquad \frac{g_2}{2} = \frac{288}{2} = 144. \tag{16}$$

Using the results from (16), we obtain the total number of RNS vectors required to find the LPS for the index g as $\frac{g_2}{2} - \frac{g_1}{2} + 1 = 144 - 122 + 1 = 23$, shown in Figure 16. The RNS vectors in Figure 16 are derived from Figure 15 by extending the $g'/2$ index from 122 to 144. The representation of the DR is now compressed from a length of 133 vectors to 23 vectors.

RNS	m_1=7	...	3	4	5	6	0	1	2	3	4	5	6	0	1	2	3	4	5	6	0	1	2	3	4	...
Vectors	m_2=8	...	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	...
$XR_{g/2}/2$	m_3=9	...	5	6	7	8	0	1	2	3	4	5	6	7	8	0	1	2	3	4	5	6	7	8	0	...
	$g/2$	...	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	...

RNS	m_1=7	...	0	1	2	3	4	5	6	0	1	2	3	4	5	6	0	1	2	3	4	5	6	0	1	...
Vectors	m_2=8	...	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	...
$XR_{g'/2}/2$	m_3=9	...	0	1	2	3	4	5	6	7	8	0	1	2	3	4	5	6	7	8	0	1	2	3	4	...
LPS	$g'/2$	...	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	...

Figure 16. RNS Vectors Spanning DR (Before and After Shifting Start Position).

This compression is possible because using only Sub-Channel 0 vectors compresses the DR by three, and exploiting its even-odd structure yields an additional compression factor of two. Thus, the total compression factor is 2*N*, and the number of RNS vectors required to find the LPS for the index g is [2]:

$$L = \left\lceil \frac{\hat{M}}{2N} \right\rceil = \left\lceil \frac{133}{6} \right\rceil = 23. \qquad (17)$$

Another useful RNS property is that subtracting a particular RNS vector from all other RNS vectors in the fundamental period shifts the start of the RNS sequence to the position of the same vector [2].

From this fact, the RNS vector $\frac{XR_{g/2}}{2} = \frac{XR_{122}}{2} = [3\ 2\ 5]^T$ is subtracted from all RNS vectors between $\frac{XR_{122}}{2}$ and $\frac{XR_{144}}{2}$ to obtain the shifted RNS sequence in the bottom row of Figure 16. This means that the solution for the index *h* is now shifted to the range $0 \le h' \le 132$ rather than $733 \le h \le 865$. This is desirable as the former can be represented in an 8-bit binary number, while the latter requires 10 bits.

3. Alignment of RNS Least Positive Solution

A method of finding the LPS to multiple systems of equations is a *positional alignment solution*. This method asserts the positions of the RNS residues for each channel of the $\frac{XR_{g'/2}}{2}$ vectors, shown in Figure 16. The LPS is the position $g'/2$ of the first $\frac{XR_{g'/2}}{2}$ vector in which all three asserted residues align [2].

After finding the LPS $g'/2$ for a particular RSNS vector $X_h = [s_1\ s_2\ s_3]^T$, the LPS is converted back to the shifted index h' by reversing the sub-channel and even-odd compression process carried out before the LPS alignment. Using F_{SC} to denote the sub-channel compensation factor, F_{odd} as the even-odd compensation factor, and (6), the expression to obtain h' is [2]:

$$h' = 3(g' + F_{odd}) + F_{SC}, \tag{18}$$

where

$$F_{odd} = \begin{cases} 0, \text{ if residue } s_1 \text{ from } X_h \text{ is even} \\ 1, \text{ if residue } s_1 \text{ from } X_h \text{ is odd} \end{cases}, \tag{19}$$

$$F_{SC} = \begin{cases} 0, \text{ if } X_h \text{ is from Sub-Channel 0} \\ 1, \text{ if } X_h \text{ is from Sub-Channel 1} \\ 2, \text{ if } X_h \text{ is from Sub-Channel 2} \end{cases}.$$

Since h' is the index h after shifting the DR start position to zero, the position of the RSNS vector within the DR, or index h, can simply be obtained by adding the DR start position h_1:

$$h = h_1 + h'. \tag{20}$$

The key principles and steps of the RSNS-to-binary conversion algorithm described in this chapter are encapsulated in Figure 17. An efficient RSNS-to-binary conversion can be achieved by exploiting the RSNS-RNS relationship, as well as using the RNS LPS and positional alignment techniques to solve for the DR position.

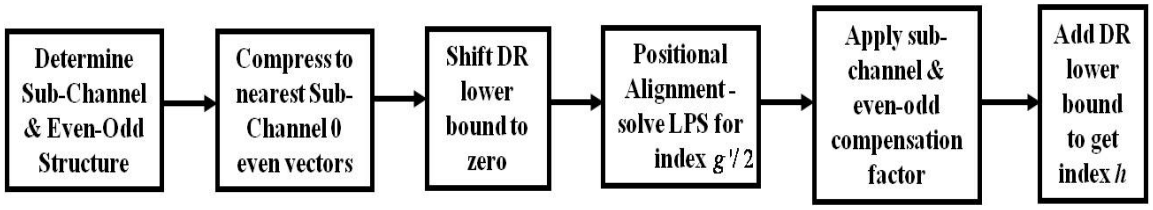

Figure 17. RSNS-to-Binary Conversion Algorithm (After [2]).

The translation of the principles of this RSNS-to-binary conversion algorithm into a feasible logic block diagram is explained in the next chapter.

III. RSNS-TO-BINARY CONVERSION

Given the key principles of the RSNS-to-binary conversion algorithm described in the last chapter, the translation of this algorithm into a feasible logic block diagram for implementation in the LabVIEW programming environment is explained in this chapter.

A. LOGIC BLOCK DIAGRAM OF RSNS-TO-BINARY CONVERTER

The overall logic block diagram of this converter that can be implemented in LabVIEW and the input/output variables for each block are shown in Figure 18. The subsequent sections will detail why each logic block is required and how it is derived.

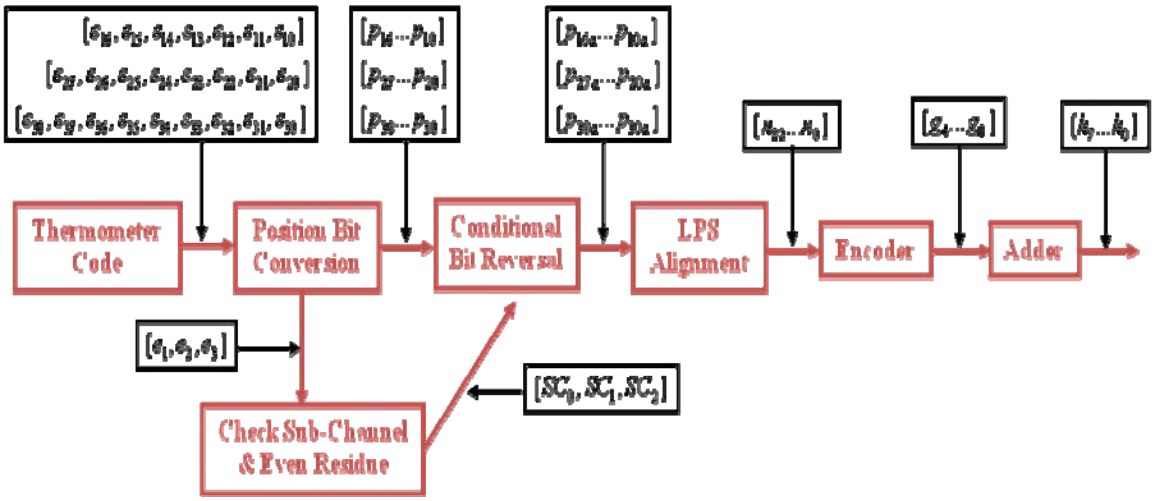

Figure 18. Logic Block Diagram of RSNS-to-Binary Converter (After [4]).

The inputs of this conversion system come from a bank of 24 comparators at the end of a photonic analog folding circuit and can be represented as a three-channel RSNS vector as shown in (2). These comparator outputs are separated into three channels of seven, eight and nine comparators, according to the Moduli $m_i = [7\ 8\ 9]$.

The input residues $[s_1, s_2, s_3]^T$ are encoded in a thermometer code for this implementation, according to (3). Each of the bits in the thermometer code is labeled as s_{ik}, where the index *i* represents the channel of the RSNS residue vector, and the index *k* is the bit position in the thermometer code, with $k = 0$ corresponding to the position of the Least Significant Bit (LSB). The RSNS thermometer code bits for $m_i = [7\ 8\ 9]$ are shown in Figure 19.

RSNS Residue Value	RSNS Thermometer Bits						
s_1	s_{16}	s_{15}	s_{14}	s_{13}	s_{12}	s_{11}	s_{10}
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1
2	0	0	0	0	0	1	1
3	0	0	0	0	1	1	1
4	0	0	0	1	1	1	1
5	0	0	1	1	1	1	1
6	0	1	1	1	1	1	1
7	1	1	1	1	1	1	1

RSNS Residue Value	RSNS Thermometer Bits							
s_2	s_{27}	s_{26}	s_{25}	s_{24}	s_{23}	s_{22}	s_{21}	s_{20}
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	1	1
3	0	0	0	0	0	1	1	1
4	0	0	0	0	1	1	1	1
5	0	0	0	1	1	1	1	1
6	0	0	1	1	1	1	1	1
7	0	1	1	1	1	1	1	1
8	1	1	1	1	1	1	1	1

RSNS Residue Value	RSNS Thermometer Bits								
s_3	s_{38}	s_{37}	s_{36}	s_{35}	s_{34}	s_{33}	s_{32}	s_{31}	s_{30}
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	1
2	0	0	0	0	0	0	0	1	1
3	0	0	0	0	0	0	1	1	1
4	0	0	0	0	0	1	1	1	1
5	0	0	0	0	1	1	1	1	1
6	0	0	0	1	1	1	1	1	1
7	0	0	1	1	1	1	1	1	1
8	0	1	1	1	1	1	1	1	1
9	1	1	1	1	1	1	1	1	1

Figure 19. RSNS Thermometer Codes for $m_i = [7\ 8\ 9]$.

B. POSITION BIT CONVERSION

The next step is to convert the RSNS thermometer code residues into RNS residues using the one-to-one correspondence property of the RSNS-RNS relationship. Each RNS residue can be represented by a unique position bit.

The position bits are denoted as p_{ik}, where the index *i* is the channel and *k* is the RNS residue value with $0 \le k \le (m_i - 1)$. The LPS is then found by finding the position $g'/2$ where the three position bits from each channel are asserted and aligned.

1. RSNS Thermometer Code to RNS Residue/Position Bit Conversion

The simplest case for this conversion is to consider the residues for Channel 1. From Equations (7), (8) and (9), we see that the top row of each set of equations is identical. This means that the conversion of RSNS residues to RNS residues for Channel 1 is the same regardless of the sub-channel that the residue comes from. The conversion process for Channel 1 $(m_1 = 7)$ is shown in Figure 20.

Step 0 ⟶ 1 ⟶ 2 ⟶ 3

RSNS Residue	**RNS Residue**		**Even RNS Residue**		**RNS Residue for PRP Moduli**	
s_1	s_1	$14 - s_1$	$\text{even}(s_1)$	$\text{even}(14 - s_1)$	$\frac{\text{even}(s_1)}{2}$	$\frac{\text{even}(14 - s_1)}{2}$
0	0	0	0	0	0	0
1	1	13	0	12	0	6
2	2	12	2	12	1	6
3	3	11	2	10	1	5
4	4	10	4	10	2	5
5	5	9	4	8	2	4
6	6	8	6	8	3	4
7	7	7	6	6	3	3

Figure 20. Channel 1 RSNS-to-RNS Conversion for all Sub-Channels (After [2]).

Recall that each RSNS residue value s_1 can be represented by two unique RNS residues to prevent ambiguity within a single channel period, as illustrated in Figure 14 previously. This conversion is shown as Step 0 to 1 in Figure 20.

The RNS residues are then rounded down to the nearest even RNS residues and divided by two to obtain RNS residues with the same PRP moduli as the RSNS, as illustrated in Figure 15 previously. This conversion is shown as Steps 1 to 3 in Figure 20. The asserted position bits corresponding to each of the RSNS and RNS residues are shown in Figure 21.

RSNS Residue	**RNS Residue for PRP Moduli**		**Position Bits**						
s_1	$\frac{\text{even}(s_1)}{2}$	$\frac{\text{even}(14-s_1)}{2}$	p_{16}	p_{15}	p_{14}	p_{13}	p_{12}	p_{11}	p_{10}
0	0	0	0	0	0	0	0	0	1
1	0	6	1	0	0	0	0	0	1
2	1	6	1	0	0	0	0	1	0
3	1	5	0	1	0	0	0	1	0
4	2	5	0	1	0	0	0	1	0
5	2	4	0	0	1	0	1	0	0
6	3	4	0	0	1	1	0	0	0
7	3	3	0	0	0	1	0	0	0

Figure 21. Channel 1 RSNS-RNS-Position Bit Correspondences for all Sub-Channels (After [2]).

The conversion process for Channel 2 $(m_2 = 8)$ and Channel 3 $(m_3 = 9)$ are similar to that of Channel 1, except that they are left-shifted by one and two positions, respectively. Examination of (7), (8) and (9) reveals that this impacts the conversion of RSNS residues from different sub-channels.

The RSNS to position bit conversion process for Sub-Channels 0 and 1 of Channel 2 are illustrated in Figures 22 and 23. The same process for Sub-Channel 2 is illustrated in Figures 24 and 25.

Note that Sub-Channel 2 is shifted by one relative to Sub-Channels 0 and 1, from (7), (8) and (9). A comparison of Figures 23 and 25 shows that the position bits of Sub-Channel 2 are the reversed of those for Sub-Channels 0 and 1.

RSNS Residue	**RNS Residue**		**Even RNS Residue**		**RNS Residue for PRP Moduli**	
s_2	s_2	$16-s_2$	$\text{even}(s_2)$	$\text{even}(16-s_2)$	$\frac{\text{even}(s_2)}{2}$	$\frac{\text{even}(16-s_2)}{2}$
0	0	0	0	0	0	0
1	1	15	0	14	0	7
2	2	14	2	14	1	7
3	3	13	2	12	1	6
4	4	12	4	12	2	6
5	5	11	4	10	2	5
6	6	10	6	10	3	5
7	7	9	6	8	3	4
8	8	8	8	8	4	4

Figure 22. Channel 2 RSNS-to-RNS Conversion for Sub-Channels 0 and 1 (After [2]).

RSNS Residue	**RNS Residue for PRP Moduli**		**Position Bits**							
s_2	$\frac{\text{even}(s_2)}{2}$	$\frac{\text{even}(16-s_2)}{2}$	p_{27}	p_{26}	p_{25}	p_{24}	p_{23}	p_{22}	p_{21}	p_{20}
0	0	0	0	0	0	0	0	0	0	1
1	0	7	1	0	0	0	0	0	0	1
2	1	7	1	0	0	0	0	0	1	0
3	1	6	0	1	0	0	0	0	1	0
4	2	6	0	1	0	0	0	1	0	0
5	2	5	0	0	1	0	0	1	0	0
6	3	5	0	0	1	0	1	0	0	0
7	3	4	0	0	0	1	1	0	0	0
8	4	4	0	0	0	1	0	0	0	0

Figure 23. Channel 2 RSNS-RNS-Position Bit Correspondences for Sub-Channels 0 and 1 (After [2]).

RSNS Residue	RNS Residue		Even RNS Residue		RNS Residue for PRP Moduli	
s_2	$s_2 - 1$	$15 - s_2$	$\text{even}(s_2 - 1)$	$\text{even}(15 - s_2)$	$\frac{\text{even}(s_2 - 1)}{2}$	$\frac{\text{even}(15 - s_2)}{2}$
0	15	15	14	14	7	7
1	0	14	0	14	0	7
2	1	13	0	12	0	6
3	2	12	2	12	1	6
4	3	11	2	10	1	5
5	4	10	4	10	2	5
6	5	9	4	8	2	4
7	6	8	6	8	3	4
8	7	7	6	6	3	3

Figure 24. Channel 2 RSNS-to-RNS Conversion for Sub-Channel 2 (After [2]).

RSNS Residue	RNS Residue for PRP Moduli		Position Bits							
s_2	$\frac{\text{even}(s_2)}{2}$	$\frac{\text{even}(15 - s_2)}{2}$	p_{27}	p_{26}	p_{25}	p_{24}	p_{23}	p_{22}	p_{21}	p_{20}
0	7	7	1	0	0	0	0	0	0	0
1	0	7	1	0	0	0	0	0	0	1
2	0	6	0	1	0	0	0	0	0	1
3	1	6	0	1	0	0	0	0	1	0
4	1	5	0	0	1	0	0	0	1	0
5	2	5	0	0	1	0	0	1	0	0
6	2	4	0	0	0	1	0	1	0	0
7	3	4	0	0	0	1	1	0	0	0
8	3	3	0	0	0	0	1	0	0	0

Figure 25. Channel 2 RSNS-RNS-Position Bit Correspondences for Sub-Channel 2 (After [2]).

The RSNS to position bit conversion process for Sub-Channel 0 of Channel 3 are illustrated in Figures 26 and 27. The same process for Sub-Channels 1 and 2 is illustrated in Figures 28 and 29. Note that Sub-Channels 1 and 2 are shifted by one relative to Sub-Channel 0, from (7), (8) and (9). A comparison of Figures 27 and 29 shows that the position bits of Sub-Channels 1 and 2 are the reversed of those for Sub-Channel 0.

RSNS Residue	**RNS Residue**		**Even RNS Residue**		**RNS Residue for PRP Moduli**	
s_3	s_3	$18-s_3$	$\text{even}(s_3)$	$\text{even}(18-s_3)$	$\frac{\text{even}(s_3)}{2}$	$\frac{\text{even}(18-s_3)}{2}$
0	0	0	0	0	0	0
1	1	17	0	16	0	8
2	2	16	2	16	1	8
3	3	15	2	14	1	7
4	4	14	4	14	2	7
5	5	13	4	12	2	6
6	6	12	6	12	3	6
7	7	11	6	10	3	5
8	8	10	8	10	4	5
9	9	9	8	8	4	4

Figure 26. Channel 3 RSNS-to-RNS Conversion for Sub-Channel 0 (After [2]).

RSNS Residue	**RNS Residue for PRP Moduli**		**Position Bits**								
s_3	$\frac{\text{even}(s_3)}{2}$	$\frac{\text{even}(18-s_3)}{2}$	p_{38}	p_{37}	p_{36}	p_{35}	p_{34}	p_{33}	p_{32}	p_{31}	p_{30}
0	0	0	0	0	0	0	0	0	0	0	1
1	0	8	1	0	0	0	0	0	0	0	1
2	1	8	1	0	0	0	0	0	0	1	0
3	1	7	0	1	0	0	0	0	0	1	0
4	2	7	0	1	0	0	0	0	1	0	0
5	2	6	0	0	1	0	0	0	1	0	0
6	3	6	0	0	1	0	0	1	0	0	0
7	3	5	0	0	0	1	0	1	0	0	0
8	4	5	0	0	0	1	1	0	0	0	0
9	4	4	0	0	0	0	1	0	0	0	0

Figure 27. Channel 3 RSNS-RNS-Position Bit Correspondences for Sub-Channel 0 (After [2]).

RSNS Residue	RNS Residue		Even RNS Residue		RNS Residue for PRP Moduli	
s_3	$s_3 - 1$	$17 - s_3$	$\text{even}(s_3)$	$\text{even}(17 - s_3)$	$\frac{\text{even}(s_3 - 1)}{2}$	$\frac{\text{even}(17 - s_3)}{2}$
0	17	17	16	16	8	8
1	0	16	0	16	0	8
2	1	15	0	14	0	7
3	2	14	2	14	1	7
4	3	13	2	12	1	6
5	4	12	4	12	2	6
6	5	11	4	10	2	5
7	6	10	6	10	3	5
8	7	9	6	8	3	4
9	8	8	8	8	4	4

Figure 28. Channel 3 RSNS-to-RNS Conversion for Sub-Channels 1 and 2 (After [2]).

RSNS Residue	RNS Residue for PRP Moduli		Position Bits								
s_3	$\frac{\text{even}(s_3 - 1)}{2}$	$\frac{\text{even}(17 - s_3)}{2}$	p_{38}	p_{37}	p_{36}	p_{35}	p_{34}	p_{33}	p_{32}	p_{31}	p_{30}
0	8	8	1	0	0	0	0	0	0	0	0
1	0	8	1	0	0	0	0	0	0	0	1
2	0	7	0	1	0	0	0	0	0	0	1
3	1	7	0	1	0	0	0	0	0	1	0
4	1	6	0	0	1	0	0	0	0	1	0
5	2	6	0	0	1	0	0	0	1	0	0
6	2	5	0	0	0	1	0	0	1	0	0
7	3	5	0	0	0	1	0	1	0	0	0
8	3	4	0	0	0	0	1	1	0	0	0
9	4	4	0	0	0	0	1	0	0	0	0

Figure 29. Channel 3 RSNS-RNS-Position Bit Correspondences for Sub-Channels 1 and 2 (After [2]).

2. Position Bit Equations

Using logic tables and Karnaugh mapping, we can show that the general equations of the position bits for even and odd moduli are [2]:

$$
\begin{aligned}
&p_{i0} = \overline{s_{i1}},\\
&p_{i1} = s_{i1}\overline{s_{i3}},\\
&p_{i2} = s_{i3}\overline{s_{i5}},\\
&\vdots\\
&p_{i\left(\left\lfloor \frac{m_i}{2} \right\rfloor -1\right)} = s_{i(m_i-3)}\overline{s_{i(m_i-1)}},\\
&p_{i\left(\left\lfloor \frac{m_i}{2} \right\rfloor\right)} = s_{i(m_i-2)},\\
&p_{i\left(\left\lfloor \frac{m_i}{2} \right\rfloor +1\right)} = s_{i(m_i-4)}\overline{s_{i(m_i-2)}},\\
&\vdots\\
&p_{i(m_i-3)} = s_{i4}\overline{s_{i6}},\\
&p_{i(m_i-2)} = s_{i2}\overline{s_{i4}},\\
&p_{i(m_i-1)} = s_{i0}\overline{s_{i2}},
\end{aligned}
\tag{21}
$$

for even moduli, and

$$
\begin{aligned}
&p_{i0} = \overline{s_{i1}},\\
&p_{i1} = s_{i1}\overline{s_{i3}},\\
&p_{i2} = s_{i3}\overline{s_{i5}},\\
&\vdots\\
&p_{i\left(\left\lfloor \frac{m_i}{2} \right\rfloor -1\right)} = s_{i(m_i-4)}\overline{s_{i(m_i-2)}},\\
&p_{i\left(\left\lfloor \frac{m_i}{2} \right\rfloor\right)} = s_{i(m_i-2)},\\
&p_{i\left(\left\lfloor \frac{m_i}{2} \right\rfloor +1\right)} = s_{i(m_i-3)}\overline{s_{i(m_i-1)}},\\
&\vdots\\
&p_{i(m_i-3)} = s_{i4}\overline{s_{i6}},\\
&p_{i(m_i-2)} = s_{i2}\overline{s_{i4}},\\
&p_{i(m_i-1)} = s_{i0}\overline{s_{i2}},
\end{aligned}
\tag{22}
$$

for odd moduli.

Equations (21) and (22) are used to generate the position bits as follows:

$$\begin{aligned} p_{10} &= \overline{s_{11}}, \\ p_{11} &= s_{11}\overline{s_{13}}, \\ p_{12} &= s_{13}\overline{s_{15}}, \\ p_{13} &= s_{15}, \\ p_{14} &= s_{14}\overline{s_{16}}, \\ p_{15} &= s_{12}\overline{s_{14}}, \\ p_{16} &= s_{10}\overline{s_{12}}, \end{aligned} \tag{23}$$

for $m_1 = 7$, and

$$\begin{aligned} p_{20} &= \overline{s_{21}}, \\ p_{21} &= s_{21}\overline{s_{23}}, \\ p_{22} &= s_{23}\overline{s_{25}}, \\ p_{23} &= s_{25}\overline{s_{27}}, \\ p_{24} &= s_{26}, \\ p_{25} &= s_{24}\overline{s_{26}}, \\ p_{26} &= s_{22}\overline{s_{24}}, \\ p_{27} &= s_{20}\overline{s_{22}}, \end{aligned} \tag{24}$$

for $m_2 = 8$, and

$$\begin{aligned} p_{30} &= \overline{s_{31}}, \\ p_{31} &= s_{31}\overline{s_{33}}, \\ p_{32} &= s_{33}\overline{s_{35}}, \\ p_{33} &= s_{35}\overline{s_{37}}, \\ p_{34} &= s_{37}, \\ p_{35} &= s_{36}\overline{s_{38}}, \\ p_{36} &= s_{34}\overline{s_{36}}, \\ p_{37} &= s_{32}\overline{s_{34}}, \\ p_{38} &= s_{30}\overline{s_{32}}, \end{aligned} \tag{25}$$

for $m_3 = 9$.

C. EVEN RESIDUE AND SUB-CHANNEL FLAGS

From Figures 20–29, it can be seen that only the position bits representing even RSNS residues for all three channels require bit reversal, depending on the sub-channel they come from. This means that a logic block is required in the converter diagram to check for even residues and their sub-channels.

1. Even Residue Flags

The letter '*e*' is used to represent an even residue flag, with the subscript denoting the channel. Each flag will be asserted when the RSNS residue is even. It was shown in [2] that the general equations of the even residue flags are:

$$\text{Even Moduli: } e_i = \overline{s_{i0}} + s_{i1}\overline{s_{i2}} + s_{i3}\overline{s_{i4}} + \cdots + s_{i(m_i-1)} \tag{26}$$

and

$$\text{Odd Moduli: } e_i = \overline{s_{i0}} + s_{i1}\overline{s_{i2}} + s_{i3}\overline{s_{i4}} + \cdots + s_{i(m_i-2)}\overline{s_{i(m_i-1)}}. \tag{27}$$

Equations (26) and (27) are used to generate the even residue flags as follows:

$$\begin{aligned} &\text{Channel 1: } e_1 = \overline{s_{10}} + s_{11}\overline{s_{12}} + s_{13}\overline{s_{14}} + s_{15}\overline{s_{16}}, \\ &\text{Channel 2: } e_2 = \overline{s_{20}} + s_{21}\overline{s_{22}} + s_{23}\overline{s_{24}} + s_{25}\overline{s_{26}} + s_{27}, \\ &\text{Channel 3: } e_3 = \overline{s_{30}} + s_{31}\overline{s_{32}} + s_{33}\overline{s_{34}} + s_{35}\overline{s_{36}} + s_{37}\overline{s_{38}}. \end{aligned} \tag{28}$$

2. Sub-Channel Flags

The symbol '*SC*' is used to represent a sub-channel flag, with the subscript denoting the channel. Each flag is used to determine if the position bits of each RSNS residue require reversal, depending on their sub-channels. With the sub-script *N* to denote the number of channels in the system, it was shown in [2] that the general equations of the sub-channel flags are:

$$
\begin{aligned}
SC_{N-1} &= e_1 \oplus e_2, \\
SC_{N-2} &= e_2 \oplus e_3, \\
&\vdots \\
SC_2 &= e_{N-2} \oplus e_{N-1}, \\
SC_1 &= e_N \oplus e_{N-1}, \\
\overline{SC_0} &= e_N \oplus e_1,
\end{aligned}
\tag{29}
$$

where $\oplus$ denotes an XOR operation.

Equation (29) is used to generate the sub-channel flags as follows:

$$
\begin{aligned}
\overline{SC_0} &= e_3 \oplus e_1, \\
SC_1 &= e_3 \oplus e_2, \\
SC_2 &= e_2 \oplus e_1.
\end{aligned}
\tag{30}
$$

D. CONDITIONAL BIT REVERSAL

Conditional bit reversal of the position bits is based on the sub-channel flags. In order to maintain proper housekeeping of variable-naming, all position bits after inversion have an additional subscript '*a*'.

Based on Figures 20–29, the position bits of Channel 1 are never reversed. The position bits of Channel 2 are reversed if the residue is from Sub-Channel 2, i.e., when SC_2 is asserted. The position bits of Channel 3 are reversed if the residue is from Sub-Channel 1 or 2, i.e., when $\overline{SC_0}$ is asserted. Note that SC_1 is not required for this shift sequence.

This reversal procedure can be accomplished via multiplexer circuits using the sub-channel flags as control signals and will be shown in Chapter IV.

E. LEAST POSITIVE SOLUTION ALIGNMENT

After determining the position bits for each channel, the position bits from all three channels are then used to compute the LPS of the positional alignment. One useful property is to make use of the one-to-one correspondence between the RNS residues and the position bits. This allows the RNS residues to be replaced by the position bits in Figure 16. This is shown in Figure 30.

Recall from (15) and (16) that the RNS vector $\frac{XR_{g/2}}{2} = \frac{XR_{122}}{2} = [3\ 2\ 5]^T$ corresponds to the RSNS vector $X_{732} = [6\ 4\ 8]^T$, which is of form $\left[e\ e\ e\right]^T$ and from Sub-Channel 0. From Figure 21, the RSNS Channel 1 residue $s_1 = 6$ corresponds to RNS residues of 3 and 4, which are asserted and highlighted in Figure 30. Similarly, from Figure 23, the RSNS Channel 2 residue $s_2 = 4$ corresponds to RNS residues of 2 and 6, and the RSNS Channel 3 residue $s_3 = 8$ corresponds to RNS residues of 3 and 5 from Figure 27.

RNS	m_1=7	...	3	4	5	6	0	1	2	3	4	5	6	0	1	2	3	4	5	6	0	1	2	3	4	...
Vectors	m_2=8	...	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	...
$XR_{g/2}/2$	m_3=9	...	5	6	7	8	0	1	2	3	4	5	6	7	8	0	1	2	3	4	5	6	7	8	0	...
LPS	$g/2$	...	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	...

Position Bits	m_1=7	...	p_{13}	p_{14}	p_{15}	p_{16}	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}	p_{16}	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}	p_{16}	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	...
	m_2=8	...	p_{22}	p_{23}	p_{24}	p_{25}	p_{26}	p_{27}	p_{20}	p_{21}	p_{22}	p_{23}	p_{24}	p_{25}	p_{26}	p_{27}	p_{20}	p_{21}	p_{22}	p_{23}	p_{24}	p_{25}	p_{26}	p_{27}	p_{20}	...
	m_3=9	...	p_{35}	p_{36}	p_{37}	p_{38}	p_{30}	p_{31}	p_{32}	p_{33}	p_{34}	p_{35}	p_{36}	p_{37}	p_{38}	p_{30}	p_{31}	p_{32}	p_{33}	p_{34}	p_{35}	p_{36}	p_{37}	p_{38}	p_{30}	...
LPS	$g'/2$	...	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	...

Figure 30. LPS Alignment using Position Bits.

The asserted RNS residues for the three channels are shown in Figure 30. Ideally, only one LPS should be active within the DR to prevent ambiguities. However, there is more than one LPS (*LPS0* and *LPS8*) asserted for this case.

There is an ambiguity in this case because the DR length (133) is not evenly divisible by six, and the DR starts at an index position (733) that is not evenly divisible by six. There are two methods to overcome this, and these are explained below.

1. Full Dynamic Range With LPS Priority

If the DR is not truncated to a length that is evenly divisible by six, it is possible that two LPS equations will be asserted at the same time – one of the boundary LPS equations (i.e., *LPS0 or LPS22*) and *LPSX*. If the full DR is to be maintained, the boundary LPS equation should always be ignored in favor of the *LPSX* solution.

This can be done by using additional logic circuitry to check for such cases and giving priority to the *LPSX* solution over the boundary LPS equations. The details of this circuitry are elaborated in Chapter V.

2. Truncated Dynamic Range

The DR can be truncated to a length that is evenly divisible by six, i.e., 126, so that it starts with a Sub-Channel 0 vector and ends with a Sub-Channel 2 vector, as required in Figure 9. The number of RNS vectors required to span the DR is now:

$$L = \left\lceil \frac{\hat{M}}{2N} \right\rceil = \left\lceil \frac{126}{6} \right\rceil = 21. \tag{31}$$

For the first case where the full DR is maintained, the number of RNS vectors required to span the DR is 23, in accordance with (17). The full 23 alignment equations are given by

$$
\begin{aligned}
&LPS0: \quad n_0 = \overline{p_{13a}\, p_{22a}\, p_{35a}},\\
&LPS1: \quad n_1 = \overline{p_{14a}\, p_{23a}\, p_{36a}},\\
&LPS2: \quad n_2 = \overline{p_{15a}\, p_{24a}\, p_{37a}},\\
&LPS3: \quad n_3 = \overline{p_{16a}\, p_{25a}\, p_{38a}},\\
&LPS4: \quad n_4 = \overline{p_{10a}\, p_{26a}\, p_{30a}},\\
&LPS5: \quad n_5 = \overline{p_{11a}\, p_{27a}\, p_{31a}},\\
&LPS6: \quad n_6 = \overline{p_{12a}\, p_{20a}\, p_{32a}},\\
&LPS7: \quad n_7 = \overline{p_{13a}\, p_{21a}\, p_{33a}},\\
&LPS8: \quad n_8 = \overline{p_{14a}\, p_{22a}\, p_{34a}},\\
&LPS9: \quad n_9 = \overline{p_{15a}\, p_{23a}\, p_{35a}},\\
&LPS10: n_{10} = \overline{p_{16a}\, p_{24a}\, p_{36a}},\\
&LPS11: n_{11} = \overline{p_{10a}\, p_{25a}\, p_{37a}},\\
&LPS12: n_{12} = \overline{p_{11a}\, p_{26a}\, p_{38a}},\\
&LPS13: n_{13} = \overline{p_{12a}\, p_{27a}\, p_{30a}},\\
&LPS14: n_{14} = \overline{p_{13a}\, p_{20a}\, p_{31a}},\\
&LPS15: n_{15} = \overline{p_{14a}\, p_{21a}\, p_{32a}},\\
&LPS16: n_{16} = \overline{p_{15a}\, p_{22a}\, p_{33a}},\\
&LPS17: n_{17} = \overline{p_{16a}\, p_{23a}\, p_{34a}},\\
&LPS18: n_{18} = \overline{p_{10a}\, p_{24a}\, p_{35a}},\\
&LPS19: n_{19} = \overline{p_{11a}\, p_{25a}\, p_{36a}},\\
&LPS20: n_{20} = \overline{p_{12a}\, p_{26a}\, p_{37a}},\\
&LPS21: n_{21} = \overline{p_{13a}\, p_{27a}\, p_{38a}},\\
&LPS22: n_{22} = \overline{p_{14a}\, p_{20a}\, p_{30a}}.
\end{aligned}
\tag{32}
$$

These operations can be accomplished using 3-input NAND gates. Note that the last two LPS equations (*LPS21* and *LPS22*) are not required for the truncated DR case.

F. ENCODER

The next step is to convert from the LPS index $g/2$ to a binary representation of the position *h* of the RSNS vector within the DR bounds. Since only one of the *LPSX* equations will be asserted at any one time, an encoder can be used to convert the 23 possible outputs to a binary representation using *B* bits, calculated via

$$B = \log_2 \lceil L \rceil. \tag{33}$$

Five bits are required for both the full and truncated DR cases, with a 23-to-5 encoder for the full DR case and a 21-to-5 encoder for the truncated DR case.

G. ADDER

As the output of the encoder is a five-bit representation of $g/2$, a left-shifted version of $g/2$ is equivalent to a multiplication by two and converts it to a six-bit index *g*. From (18), the F_{odd} compensation factor has to be computed next, as the DR was compressed using even residues only. The factor F_{odd} is equal to $\overline{e_1}$ since the complement of e_1 is asserted when the Channel 1 residue s_1 is odd, as shown in (28). The LSB of *g* is guaranteed to be zero, as *g* is a left-shifted version of $g/2$. Thus, the even residue flag $\overline{e}_1$ can replace the LSB of *g*.

From Equation (18), the computation of $3(g+\overline{e}_1)$ is also required. It is easier to implement this as $3X = 2X + X$ in hardware using a wired shift and an adder rather than a multiplication by three using a multiplexer. To achieve this, $(g+\overline{e}_1)$ is shifted one position left to form $2\left(g+\overline{e}_1\right)$ and used as the input to an eight-bit carry-look-ahead adder. This concept is illustrated in Figure 31, where the notation g_b refers to the b^{th} bit of the binary representation of index *g* [2].

$$
\begin{array}{lcccl}
2X & & 2\left(g+\overline{e_1}\right) & & \quad g_4\ \ g_3\ \ g_2\ \ g_1\ \ g_0\ \ \overline{e_1}\ \ 0 \\
\underline{X+} & \rightarrow & \left(g+\overline{e_1}\right)+ & \rightarrow & \underline{\quad\quad g_4\ \ g_3\ \ g_2\ \ g_1\ \ g_0\ \ \overline{e_1}\ +} \\
3X & & 3\left(g+\overline{e_1}\right) & & h_7'\ \ h_6'\ \ h_5'\ \ h_4'\ \ h_3'\ \ h_2'\ \ h_1'\ \ h_0'
\end{array}
$$

Figure 31. Adder Function for Implementing Multiplication by Three (After [2]).

Lastly, Equation (18) requires the addition of the F_{SC} compensation factor, as the DR was compressed using Sub-Channel 0 vectors only. The F_{SC} compensation factor is represented by the signals $SC2$ and $\overline{SC0}$ from (30), which are asserted if the Channel 2 residue is from Sub-Channel 2, and the Channel 3 residue is from Sub-Channel 1 or 2, respectively.

Fortunately, the left shift of $(g+\overline{e}_1)$ provides a LSB slot guaranteed to be zero, and the carry-in to the adder provides another LSB slot. This allows $(g+\overline{e}_1)$, $2\left(g+\overline{e}_1\right)$, $\overline{SC0}$, and $SC2$ to be summed in a single adder [2], shown in Figure 32.

$$
\begin{array}{lcccl}
2X & & 2\left(g+\overline{e_1}\right) & & \quad g_4\ \ g_3\ \ g_2\ \ g_1\ \ g_0\ \ \overline{e_1}\ \ \overline{SC_0} \leftarrow SC_2 \text{ (carry-in)} \\
\underline{X+} & \rightarrow & \left(g+\overline{e_1}\right)+ & \rightarrow & \underline{\quad\quad g_4\ \ g_3\ \ g_2\ \ g_1\ \ g_0\ \ \overline{e_1}\ +} \\
3X & & 3\left(g+\overline{e_1}\right) & & h_7'\ \ h_6'\ \ h_5'\ \ h_4'\ \ h_3'\ \ h_2'\ \ h_1'\ \ h_0'
\end{array}
$$

Figure 32. Single Adder for Converting LPS to Binary (After [2]).

The output of this adder is the binary representation of the position *h* within the DR for the RSNS vector $X_h = \left[s_1 \quad s_2 \quad s_3\right]^{\mathrm{T}}$. The LabVIEW circuit schematics that implement the logic equations developed in this chapter are provided in the next chapter.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. LABVIEW IMPLEMENTATION OF RSNS-TO-BINARY CONVERSION

Using the logic equations developed in the previous chapter, we document the process of implementing the RSNS-to-binary converter in the NI LabVIEW programming environment in this chapter.

A. LABVIEW SCHEMATICS OF RSNS-TO-BINARY CONVERTER

The NI LabVIEW programming environment was specifically selected as its programming structure is modular in nature. This allows the codes for each logic block to be stored as sub-routines or sub-Virtual Instruments (sub-VIs) and run as part of a larger routine or Virtual Instrument (VI).

This modular structure is beneficial for the future expansion of such RSNS ADC system as it allows additional sub-routines to be designed and added to existing routines should it be decided to scale the ADC to higher moduli configurations in future.

In addition, if the ADC requires future upgrading, it is relatively easy to replace current FPGA modules with other higher-speed and higher-bandwidth NI modules with minimal disruption to existing codes due to its plug-and-play features.

The overall schematics of this converter in LabVIEW are shown in Figure 33 based on the major logic blocks developed in Chapter III. The process of constructing each logic block as a sub-routine and implementing it as part of the overall routine is detailed in the subsequent sections.

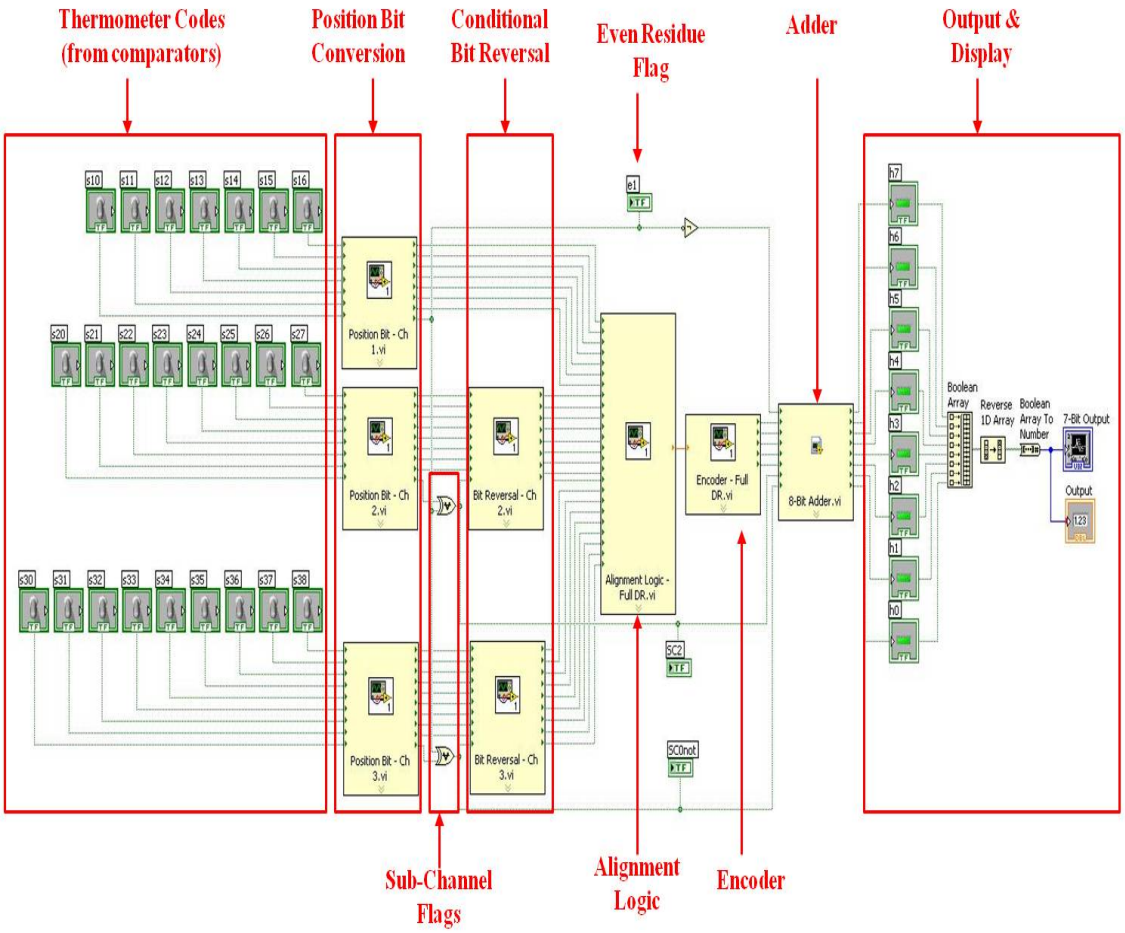

Figure 33. Overall LabVIEW Schematics of RSNS-to-Binary Converter.

B. POSITION BIT CONVERSION AND EVEN RESIDUE/SUB-CHANNEL FLAGS

The logic blocks for position bit conversion and even residue flags are combined, as both of them can be calculated using the thermometer codes from the comparators. The Boolean equations used are modified from those in Chapter III. The only changes are the application of DeMorgan's Theorem to use NAND, NOR and inverter gates predominantly.

1. Channel 1

DeMorgan's Theorem was applied to equations (23) and (28) to obtain:

$$
\begin{aligned}
p_{10} &= \overline{s_{11}}, \\
p_{11} &= s_{11}\overline{s_{13}} = \overline{\overline{s_{11}} + s_{13}}, \\
p_{12} &= s_{13}\overline{s_{15}} = \overline{\overline{s_{13}} + s_{15}}, \\
p_{13} &= s_{15}, \\
p_{14} &= s_{14}\overline{s_{16}} = \overline{\overline{s_{14}} + s_{16}}, \\
p_{15} &= s_{12}\overline{s_{14}} = \overline{\overline{s_{12}} + s_{14}}, \\
p_{16} &= s_{10}\overline{s_{12}} = \overline{\overline{s_{10}} + s_{12}},
\end{aligned}
\tag{34}
$$

and

$$
e_1 = \overline{s_{10}} + s_{11}\overline{s_{12}} + s_{13}\overline{s_{14}} + s_{15}\overline{s_{16}} = \overline{s_{10}\left(\overline{s_{11}\,\overline{s_{12}}}\right)\left(\overline{s_{13}\,\overline{s_{14}}}\right)\left(\overline{s_{15}\,\overline{s_{16}}}\right)}. \tag{35}
$$

The implementation of equations (34) and (35) in LabVIEW is shown in Figure 33.

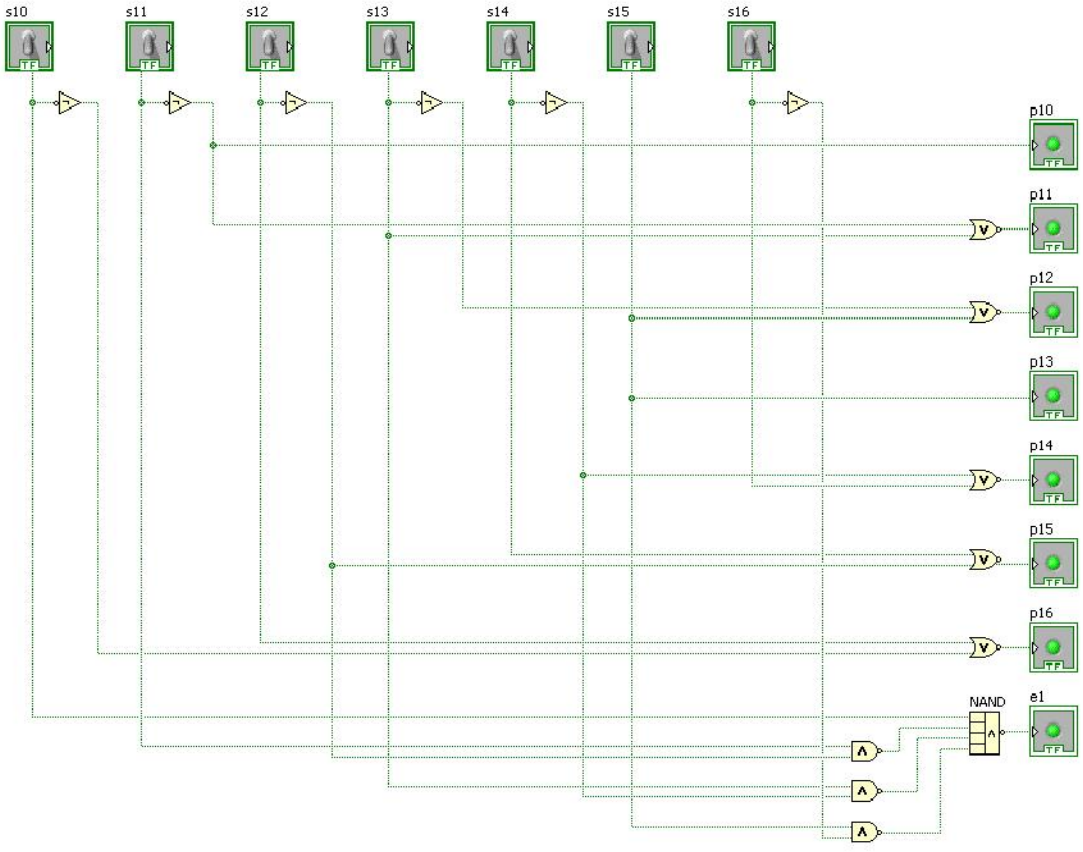

Figure 34. LabVIEW Schematics of Channel 1 Position Bit and Even Residue Flag.

2. Channel 2

DeMorgan's Theorem was applied to equations (24) and (28) to obtain

$$\begin{aligned}
p_{20} &= \overline{s_{21}}, \\
p_{21} &= s_{21}\overline{s_{23}} = \overline{\overline{s_{21}} + s_{23}}, \\
p_{22} &= s_{23}\overline{s_{25}} = \overline{\overline{s_{23}} + s_{25}}, \\
p_{23} &= s_{25}\overline{s_{27}} = \overline{\overline{s_{25}} + s_{27}}, \\
p_{24} &= s_{26}, \\
p_{25} &= s_{24}\overline{s_{26}} = \overline{\overline{s_{24}} + s_{26}}, \\
p_{26} &= s_{22}\overline{s_{24}} = \overline{\overline{s_{22}} + s_{24}}, \\
p_{27} &= s_{20}\overline{s_{22}} = \overline{\overline{s_{20}} + s_{22}},
\end{aligned} \tag{36}$$

and

$$e_2 = \overline{s_{20}} + s_{21}\overline{s_{22}} + s_{23}\overline{s_{24}} + s_{25}\overline{s_{26}} + s_{27} = \overline{s_{20}\left(s_{21}\overline{s_{22}}\right)\left(s_{23}\overline{s_{24}}\right)\left(s_{25}\overline{s_{26}}\right)\overline{s_{27}}}. \tag{37}$$

The implementation of equations (36) and (37) in LabVIEW is shown in Figure 35.

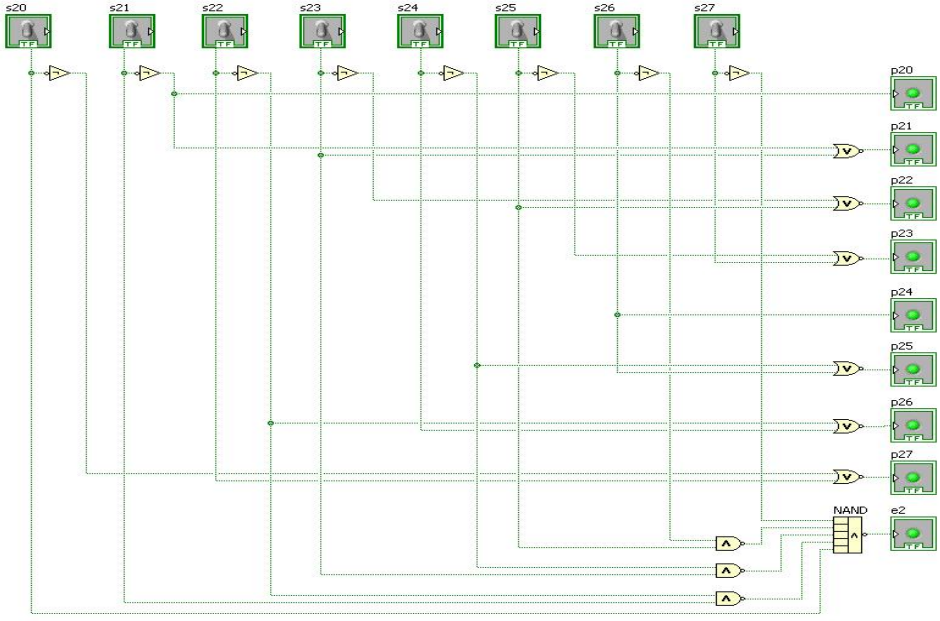

Figure 35. LabVIEW Schematics of Channel 2 Position Bit and Even Residue Flag.

3. Channel 3

DeMorgan's Theorem was applied to equations (25) and (28) to obtain

$$\begin{aligned}
p_{30} &= \overline{s_{31}}, \\
p_{31} &= s_{31}\overline{s_{33}} = \overline{\overline{s_{31}} + s_{33}}, \\
p_{32} &= s_{33}\overline{s_{35}} = \overline{\overline{s_{33}} + s_{35}}, \\
p_{33} &= s_{35}\overline{s_{37}} = \overline{\overline{s_{35}} + s_{37}}, \\
p_{34} &= s_{37}, \\
p_{35} &= s_{36}\overline{s_{38}} = \overline{\overline{s_{36}} + s_{38}}, \\
p_{36} &= s_{34}\overline{s_{36}} = \overline{\overline{s_{34}} + s_{36}}, \\
p_{37} &= s_{32}\overline{s_{34}} = \overline{\overline{s_{32}} + s_{34}}, \\
p_{38} &= s_{30}\overline{s_{32}} = \overline{\overline{s_{30}} + s_{32}},
\end{aligned} \tag{38}$$

and

$$e_3 = \overline{s_{30}} + s_{31}\overline{s_{32}} + s_{33}\overline{s_{34}} + s_{35}\overline{s_{36}} + s_{37}\overline{s_{38}} = \overline{s_{30}\overline{\left(s_{31}\overline{s_{32}}\right)}\,\overline{\left(s_{33}\overline{s_{34}}\right)}\,\overline{\left(s_{35}\overline{s_{36}}\right)}\,\overline{\left(s_{37}\overline{s_{38}}\right)}}. \tag{39}$$

The implementation of equations (38) and (39) in LabVIEW is shown in Figure 36.

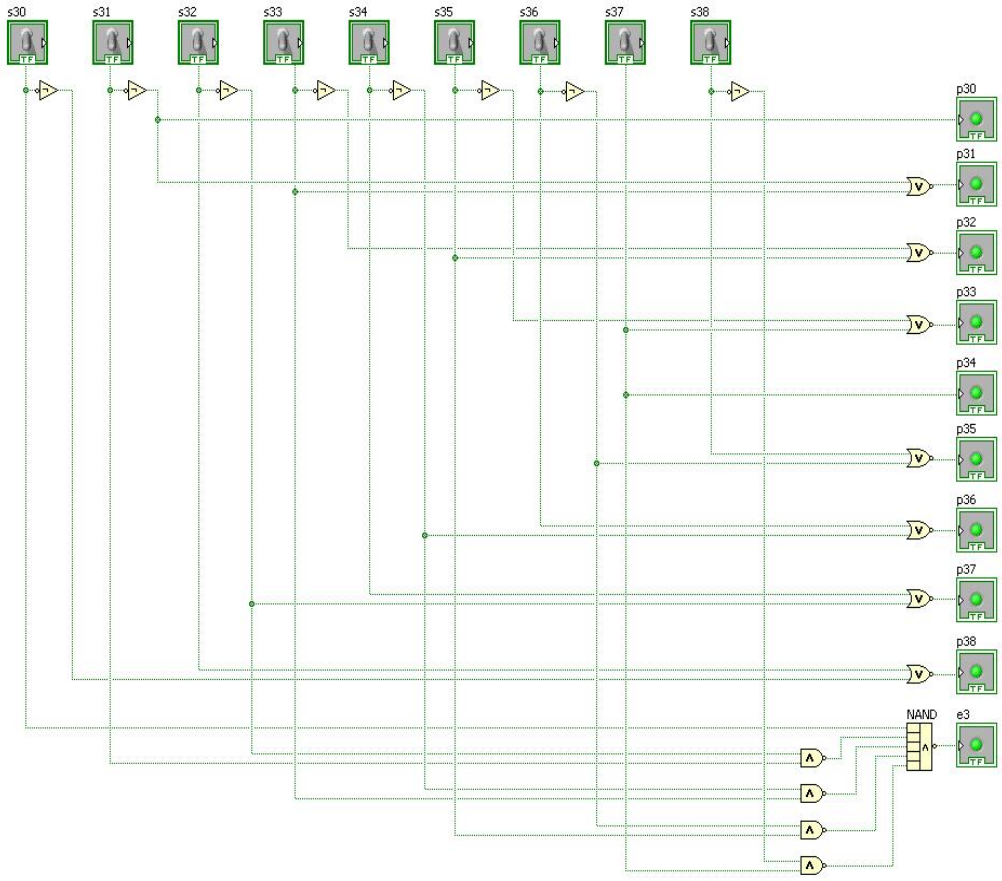

Figure 36. LabVIEW Schematics of Channel 3 Position Bit and Even Residue Flag.

4. Sub-Channel Flags

Equation (30) was used to construct the Sub-Channel flags in Figures 37 and 38 using XOR gates. They are used to check if the position bits of each RSNS residue require bit reversal, depending on which sub-channel it is from.

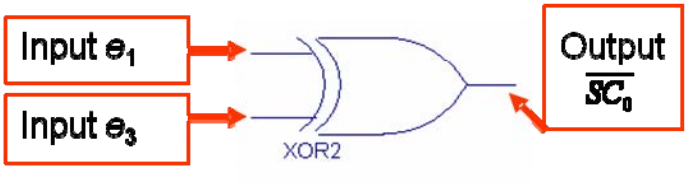

Figure 37. Sub-Channel 0 Flag.

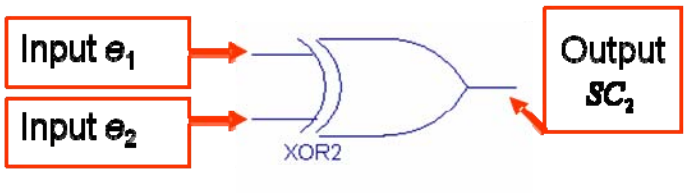

Figure 38. Sub-Channel 2 Flag.

C. CONDITIONAL BIT REVERSAL

Each inverted position bit was mapped from two possible states, depending on whether it was inverted or not. For example, the position bit p_{20a} was mapped from p_{20} if it was not inverted and mapped from p_{27} if it was inverted. Thus, the Boolean expression for position bit p_{20a} can be expressed as: $p_{20a} = p_{27}SC_2 + p_{20}\overline{SC_2} = \overline{\left(\overline{p_{27}SC_2}\right)\left(\overline{p_{20}\overline{SC_2}}\right)}$.

The conditional bit reversal for each position bit can be accomplished by the use of a 2-to-1 multiplexer, shown in Figure 39, using the sub-channel flag as a control signal. The desired output is either Input 1 or Input 2, depending on whether the control signal is asserted or not asserted, respectively. This procedure can be applied to all the other position bits.

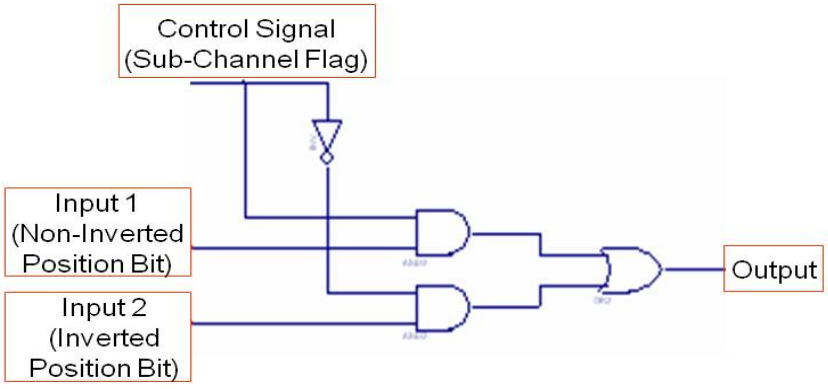

Figure 39. 2-to-1 Multiplexer (After [4]).

Given Figure 39 as a basic building block, the multiplexer architecture is extended to form the Channel 2 and Channel 3 bit reversal circuits, shown in the next two sections. The only modifications are that NAND gates are predominantly used to achieve the same logic operation as the AND and OR gates in Figure 39.

1. Channel 2 Reversal

Recall that the position bits of Channel 1 are never reversed from Figure 21. Thus, a Channel 1 reversal circuit is not required for this implementation. For Channel 2, the position bits are reversed if the residue is from Sub-Channel 2, i.e., when SC_2 is asserted, from a comparison of Figures 23 and 25. The LabVIEW schematics of the Channel 2 conditional bit reversal circuit are shown in Figure 40.

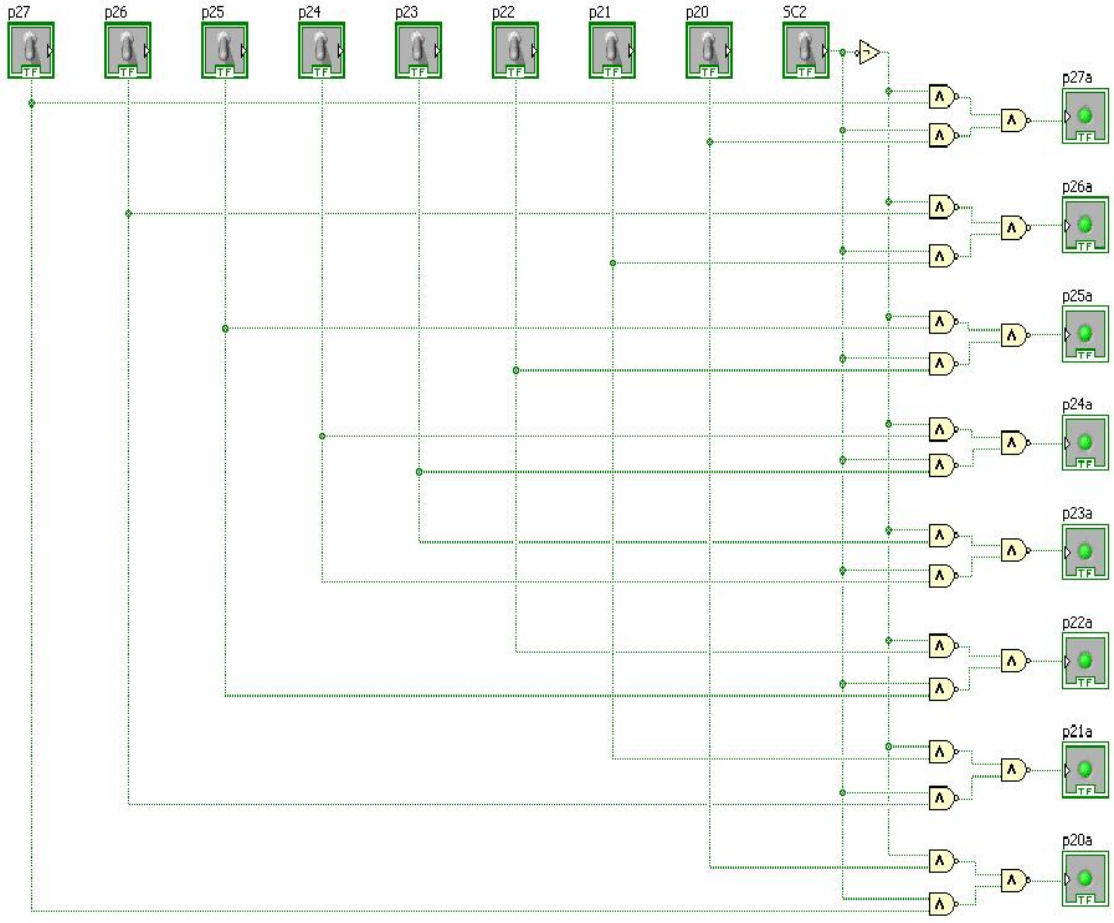

Figure 40. LabVIEW Schematics of Channel 2 Conditional Bit Reversal.

2. Channel 3 Reversal

For Channel 3, the position bits are reversed if the residue is from Sub-Channel 1 or 2, i.e., when $\overline{SC_0}$ is asserted, from a comparison of Figures 27 and 29. The LabVIEW schematics of the Channel 3 conditional bit reversal circuit are shown in Figure 41.

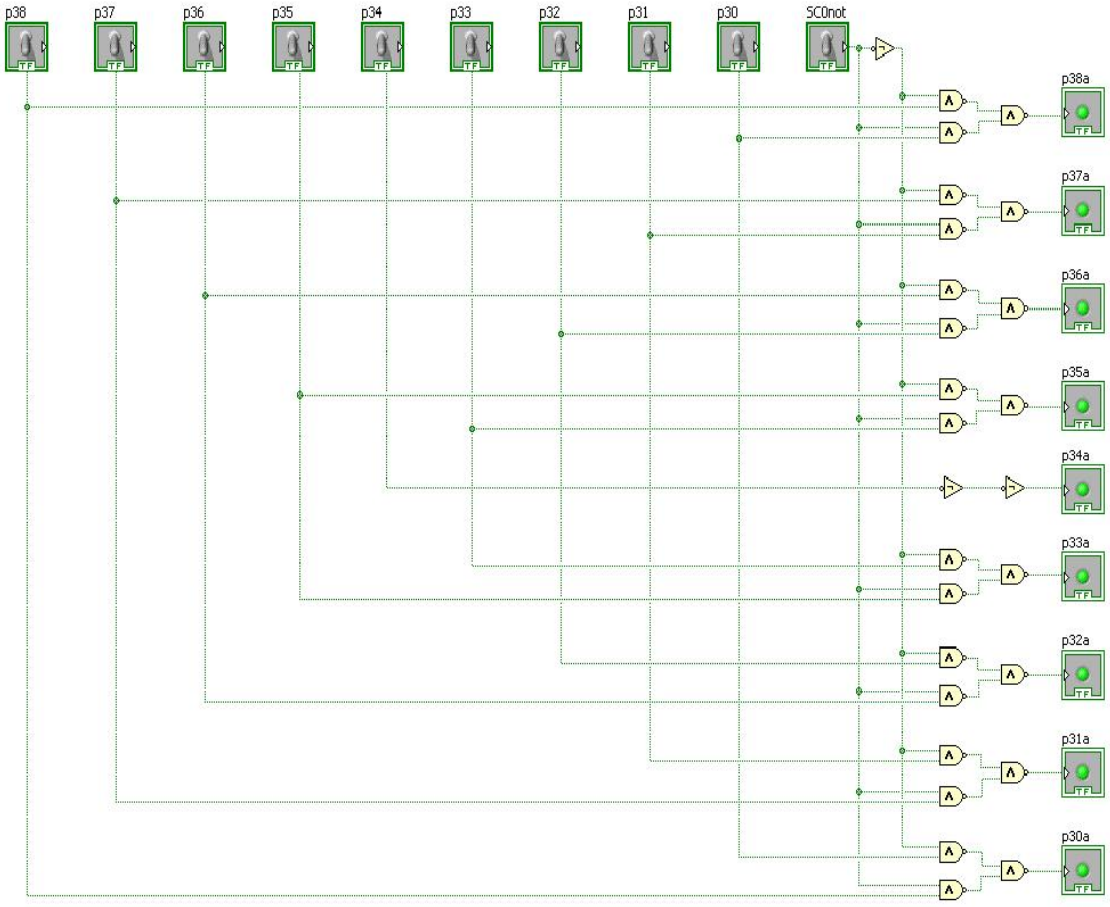

Figure 41. LabVIEW Schematics of Channel 3 Conditional Bit Reversal.

D. ALIGNMENT LOGIC

The LabVIEW schematics of the alignment logic circuit to achieve the full DR of 133 are shown in Figure 42. It is a direct mapping of equation (32). The only modification is the insertion of inverters to allow the proper functioning of the encoder circuit.

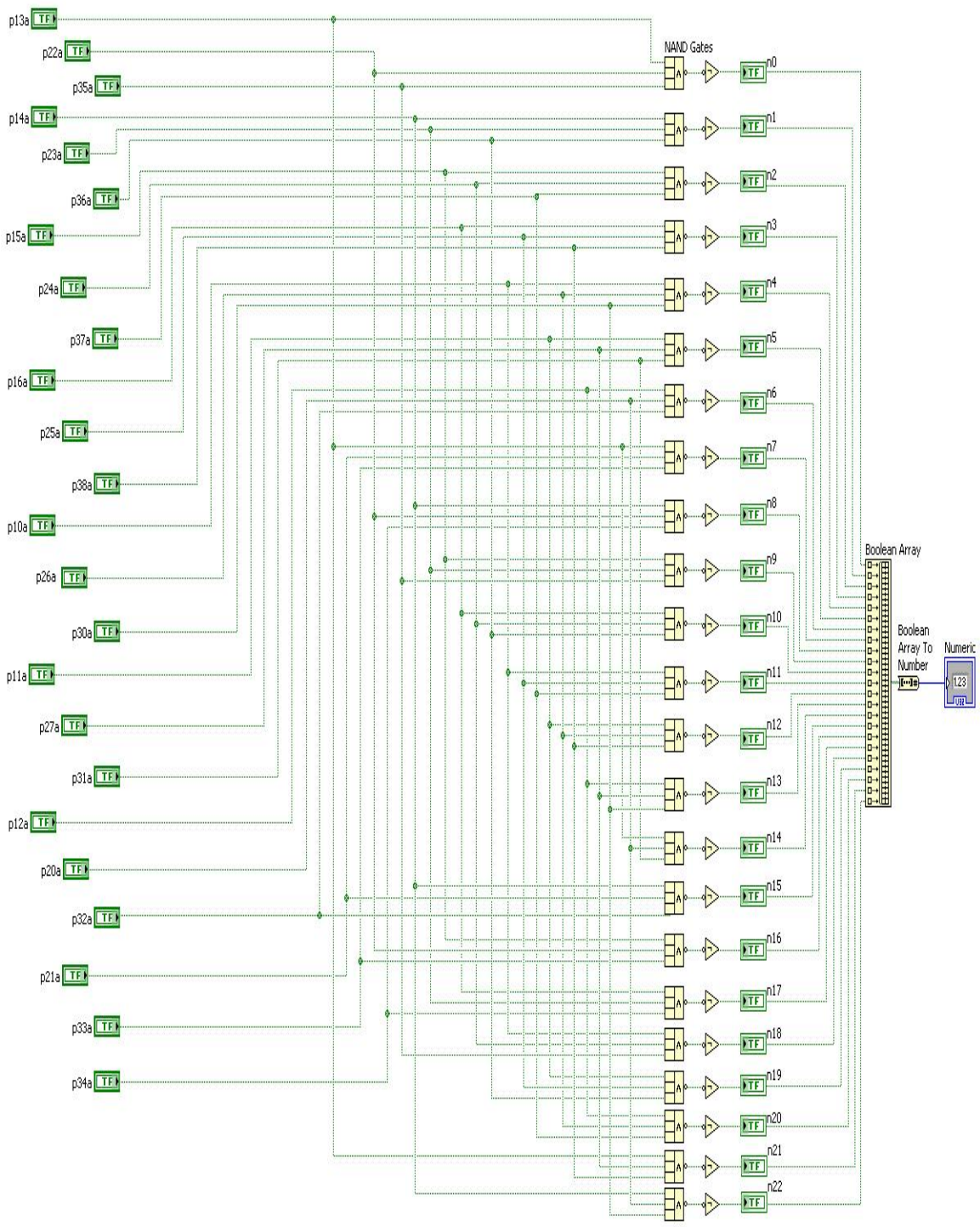

Figure 42. LabVIEW Schematics of Alignment Logic (Full DR).

Note that the last two LPS equations (*LPS21* and *LPS22*) are not required for the truncated DR case, and have to be removed if it is implemented.

E. ENCODER

Since only one of the *LPSX* equations will be asserted at any one time in the Alignment Logic circuit in Figure 42, an encoder can be used to convert the 23 possible outputs to a 5-bit representation.

To achieve this, a logic table is derived in Table 2. The function of the encoder is to convert the LPS number of the asserted NAND gate into a five-bit output. For example, if n22 is the active NAND gate, the encoder output will be 10110.

Table 2. Encoder Logic Table.

Active NAND Gate	g4	g3	g2	g1	g0
n0	0	0	0	0	0
n1	0	0	0	0	1
n2	0	0	0	1	0
n3	0	0	0	1	1
n4	0	0	1	0	0
n5	0	0	1	0	1
n6	0	0	1	1	0
n7	0	0	1	1	1
n8	0	1	0	0	0
n9	0	1	0	0	1
n10	0	1	0	1	0
n11	0	1	0	1	1
n12	0	1	1	0	0
n13	0	1	1	0	1
n14	0	1	1	1	0
n15	0	1	1	1	1
n16	1	0	0	0	0
n17	1	0	0	0	1
n18	1	0	0	1	0
n19	1	0	0	1	1
n20	1	0	1	0	0
n21	1	0	1	0	1
n22	1	0	1	1	0

The output bits of the encoder are thus the OR combination of all the logic 1s in their respective columns in Table 2. For example, g4 will be asserted if any of the n16, n17, n18, n19, n20, n21 and n22 gate is active. Again, note that the last two gates are not required for the truncated DR case and have to be removed if it is implemented.

The LabVIEW implementation of the encoder for the truncated DR and the full DR cases are highlighted in the next two sections.

1. Truncated Dynamic Range

The logical binary expressions for the 21-to-5 encoder for the truncated DR case, derived from Table 2, are given by

$$
\begin{aligned}
g_0 &= \overline{(n_1 + n_3 + n_5)(n_7 + n_9 + n_{11})(n_{13} + n_{15})(n_{17} + n_{19})} \\
g_1 &= \overline{(n_2 + n_3 + n_6)(n_7 + n_{10} + n_{11})(n_{14} + n_{15})(n_{18} + n_{19})} \\
g_2 &= \overline{(n_4 + n_5 + n_6)(n_7 + n_{12} + n_{13})(n_{14} + n_{15} + n_{20})} \\
g_3 &= \overline{(n_8 + n_9 + n_{10})(n_{11} + n_{12} + n_{13})(n_{14} + n_{15})} \\
g_4 &= \overline{(n_{16} + n_{17} + n_{18})(n_{19} + n_{20})}.
\end{aligned}
\tag{40}
$$

The implementation of equation (40) in LabVIEW as an encoder circuit to achieve the truncated DR of 128 is shown in Figure 43.

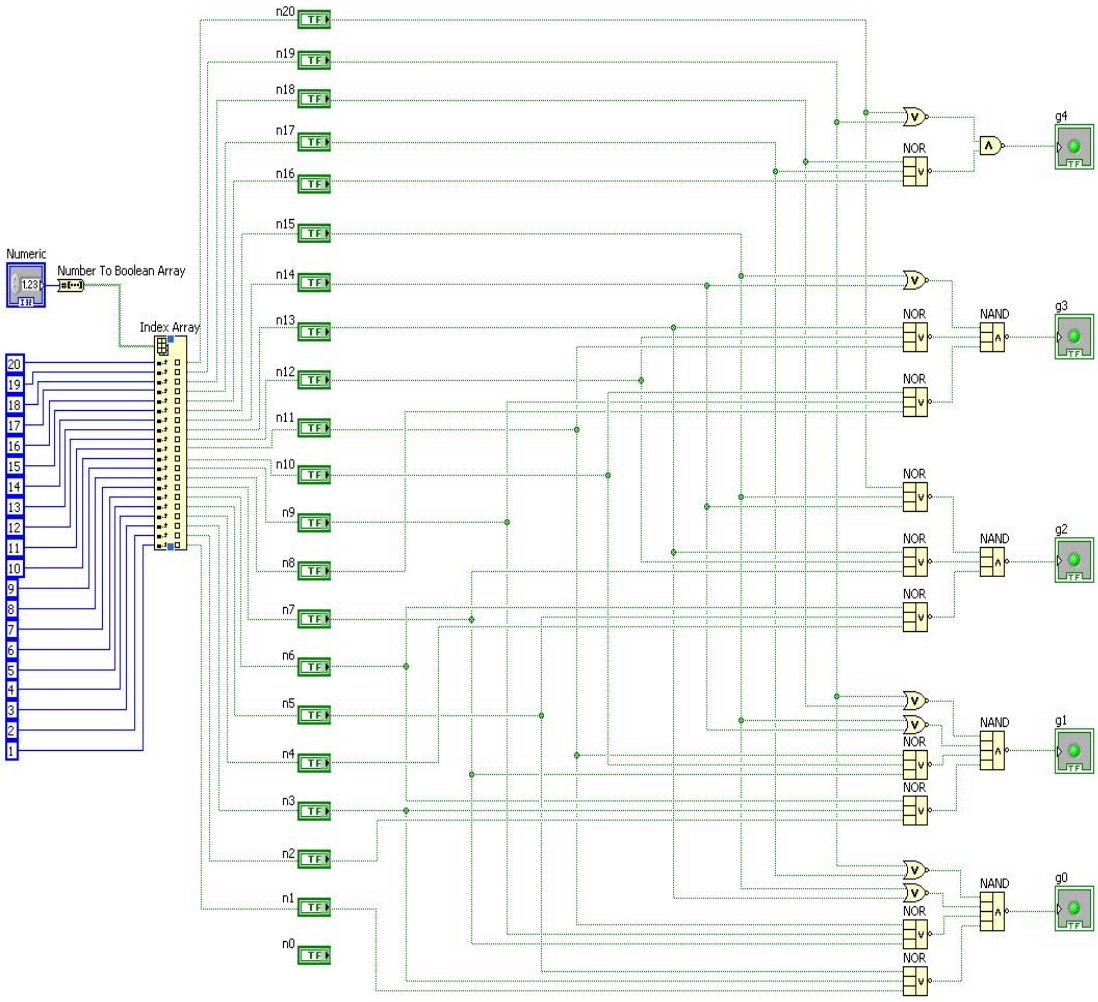

Figure 43. LabVIEW Schematics of Encoder (Truncated DR).

2. Full Dynamic Range

The logical binary expressions for the 23-to-5 encoder for the full DR case, derived from Table 2, are given by

$$
\begin{aligned}
g_0 &= \overline{(n_1+n_3+n_5)(n_7+n_9+n_{11})(n_{13}+n_{15}+n_{17})(n_{19}+n_{21})} \\
g_1 &= \overline{(n_2+n_3+n_6)(n_7+n_{10}+n_{11})(n_{14}+n_{15}+n_{18})(n_{19}+n_{22})} \\
g_2 &= \overline{(n_4+n_5+n_6)(n_7+n_{12}+n_{13})(n_{14}+n_{15}+n_{20})(n_{21}+n_{22})} \\
g_3 &= \overline{(n_8+n_9+n_{10})(n_{11}+n_{12}+n_{13})(n_{14}+n_{15})} \\
g_4 &= \overline{(n_{16}+n_{17}+n_{18})(n_{19}+n_{20})(n_{21}+n_{22})}.
\end{aligned}
\tag{41}
$$

The implementation of equation (41) in LabVIEW as an encoder circuit is shown in Figure 44.

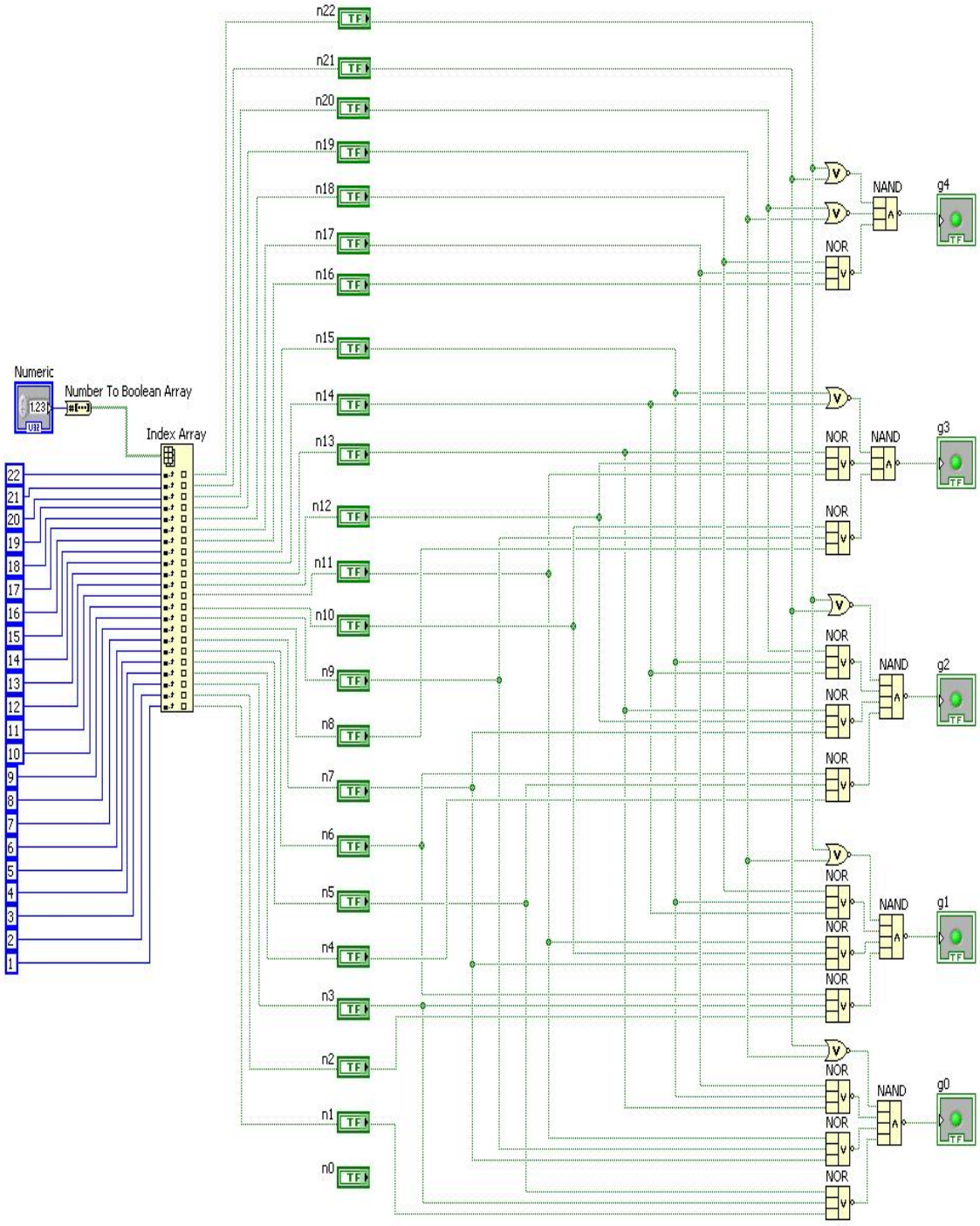

Figure 44. LabVIEW Schematics of Encoder (Full DR).

F. ADDER

The LabVIEW schematics of the adder circuit to convert the encoder's 5-bit representation to a binary representation of the position *h* within the DR are shown in Figure 45. It is a direct implementation of Figure 32.

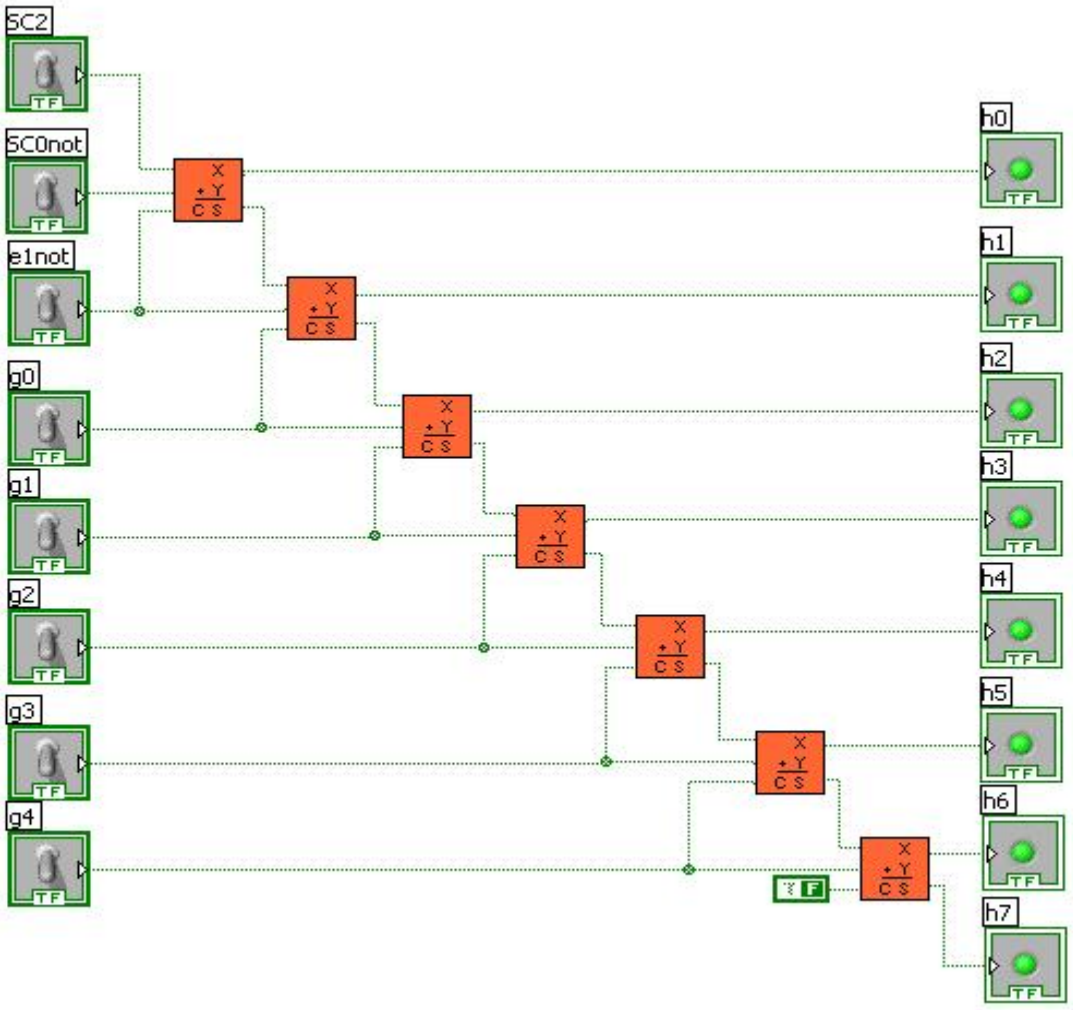

Figure 45. LabVIEW Schematics of Adder.

The implementation of the logic blocks of the RSNS-to-binary converter in LabVIEW was shown in this chapter. The simulation and testing of this converter to verify that the conversion logic is functioning properly is documented in the next chapter.

THIS PAGE INTENTIONALLY LEFT BLANK

V. SIMULATION OF RSNS-TO-BINARY CONVERSION

The simulation and testing of the RSNS-to-binary converter to verify that the conversion logic is functioning properly is contained in this chapter. The first section of this chapter is a description of the creation of thermometer codes for each channel in LabVIEW to simulate as the inputs to the conversion algorithm. The second section contains the key results obtained for three simulation models (truncated DR, full DR, and full DR with LPS Priority cases).

A. LABVIEW THERMOMETER CODE GENERATORS

The LabVIEW software does not have functions to generate the thermometer codes directly. However, the one-to-one correspondence between the RNS and RSNS vectors can be exploited to convert a RNS structure into a RSNS thermometer code.

Previously, it was demonstrated in Figure 14 that each RSNS thermometer code can be mapped to a unique RNS state such that there is no ambiguity within a single channel period. Thus, LabVIEW functions can be created to act as counters to cycle through the RNS states for each channel, and digital logic can be added to map each RNS state to the required RSNS thermometer code. The next three sections contain the details of the procedure to generate the thermometer codes for each of the three channels.

1. Channel 1

The logic table for Channel 1 is shown in Table 3. The RNS states are represented by the bits $\{i_{13}, i_{12}, i_{11}, i_{10}\}$ and are the inputs to the Channel 1 thermometer code generator. The desired thermometer code output is represented by the RSNS bits $\{s_{16}, s_{15}, s_{14}, s_{13}, s_{12}, s_{11}, s_{10}\}$.

Karnaugh maps are used to carry out logic minimization for Table 3, shown in Figure 46.

Table 3. Channel 1 ($m_1 = 7$) Logic Table.

RNS State	i_{13}	i_{12}	i_{11}	i_{10}	RSNS State	s_{16}	s_{15}	s_{14}	s_{13}	s_{12}	s_{11}	s_{10}
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	1	0	0	0	0	0	0	1
2	0	0	1	0	2	0	0	0	0	0	1	1
3	0	0	1	1	3	0	0	0	0	1	1	1
4	0	1	0	0	4	0	0	0	1	1	1	1
5	0	1	0	1	5	0	0	1	1	1	1	1
6	0	1	1	0	6	0	1	1	1	1	1	1
7	0	1	1	1	7	1	1	1	1	1	1	1
8	1	0	0	0	6	0	1	1	1	1	1	1
9	1	0	0	1	5	0	0	1	1	1	1	1
10	1	0	1	0	4	0	0	0	1	1	1	1
11	1	0	1	1	3	0	0	0	0	1	1	1
12	1	1	0	0	2	0	0	0	0	0	1	1
13	1	1	0	1	1	0	0	0	0	0	0	1

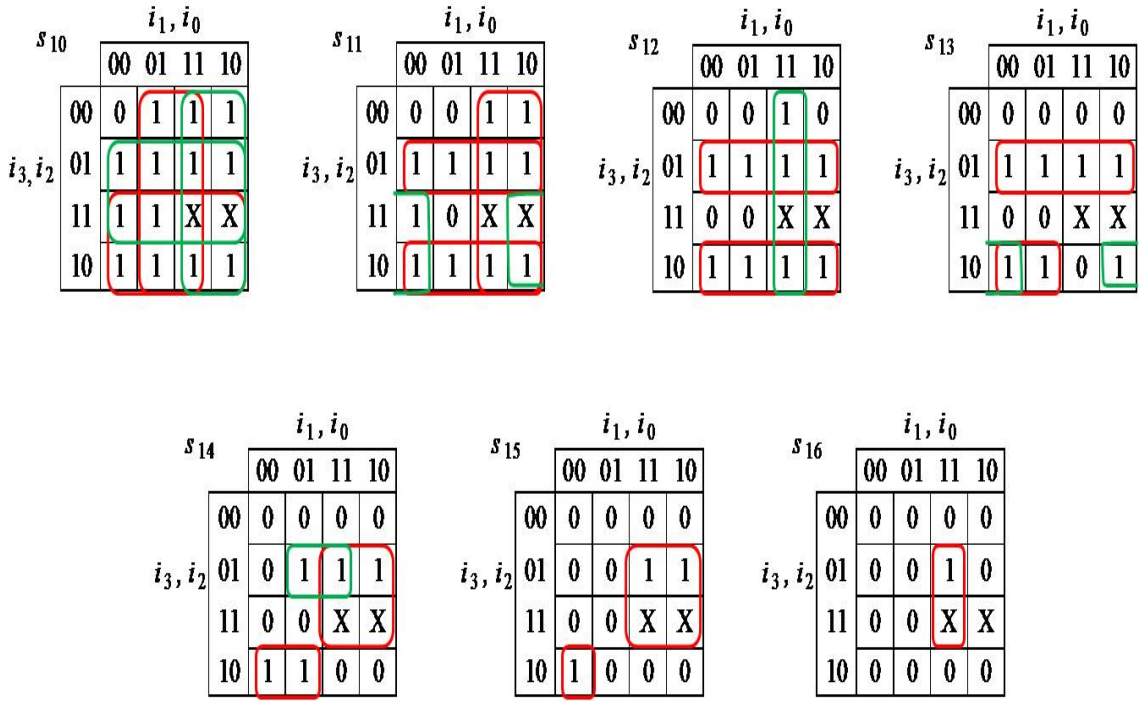

Figure 46. Channel 1 Logic Minimization of Table 3 (After [4]).

The logic minimization in Figure 46 is used to produce the logic equations

$$
\begin{aligned}
s_{10} &= i_3 + i_2 + i_1 + i_0 = \overline{\overline{i_3 + i_2 + i_1 + i_0}} \\
s_{11} &= i_1 + \overline{i}_3 i_2 + i_3 \overline{i}_2 + i_3 \overline{i}_0 = \overline{\overline{i}_1 \left(\overline{\overline{i}_3 i_2}\right)\left(\overline{i_3 \overline{i}_2}\right)\left(\overline{i_3 \overline{i}_0}\right)} \\
s_{12} &= \overline{i}_3 i_2 + i_3 \overline{i}_2 + i_1 i_0 = \overline{\left(\overline{\overline{i}_3 i_2}\right)\left(\overline{i_3 \overline{i}_2}\right)\left(\overline{i_1 i_0}\right)} \\
s_{13} &= \overline{i}_3 i_2 + i_3 \overline{i}_2 \overline{i}_1 + i_3 \overline{i}_2 \overline{i}_0 = \overline{\left(\overline{\overline{i}_3 i_2}\right)\left(\overline{i_3 \overline{i}_2 \overline{i}_1}\right)\left(\overline{i_3 \overline{i}_2 \overline{i}_0}\right)} \\
s_{14} &= i_2 i_1 + \overline{i}_3 i_2 i_0 + i_3 \overline{i}_2 \overline{i}_1 = \overline{\left(\overline{i_2 i_1}\right)\left(\overline{\overline{i}_3 i_2 i_0}\right)\left(\overline{i_3 \overline{i}_2 \overline{i}_1}\right)} \\
s_{15} &= i_2 i_1 + i_3 \overline{i}_2 \overline{i}_1 \overline{i}_0 = \overline{\left(\overline{i_2 i_1}\right)\left(\overline{i_3 \overline{i}_2 \overline{i}_1 \overline{i}_0}\right)} \\
s_{16} &= i_2 i_1 i_0 = \overline{i_2 + i_1 + i_0}
\end{aligned}
\tag{42}
$$

These logic equations are used to implement the Channel 1 thermometer code generator in LabVIEW, shown in Figure 47.

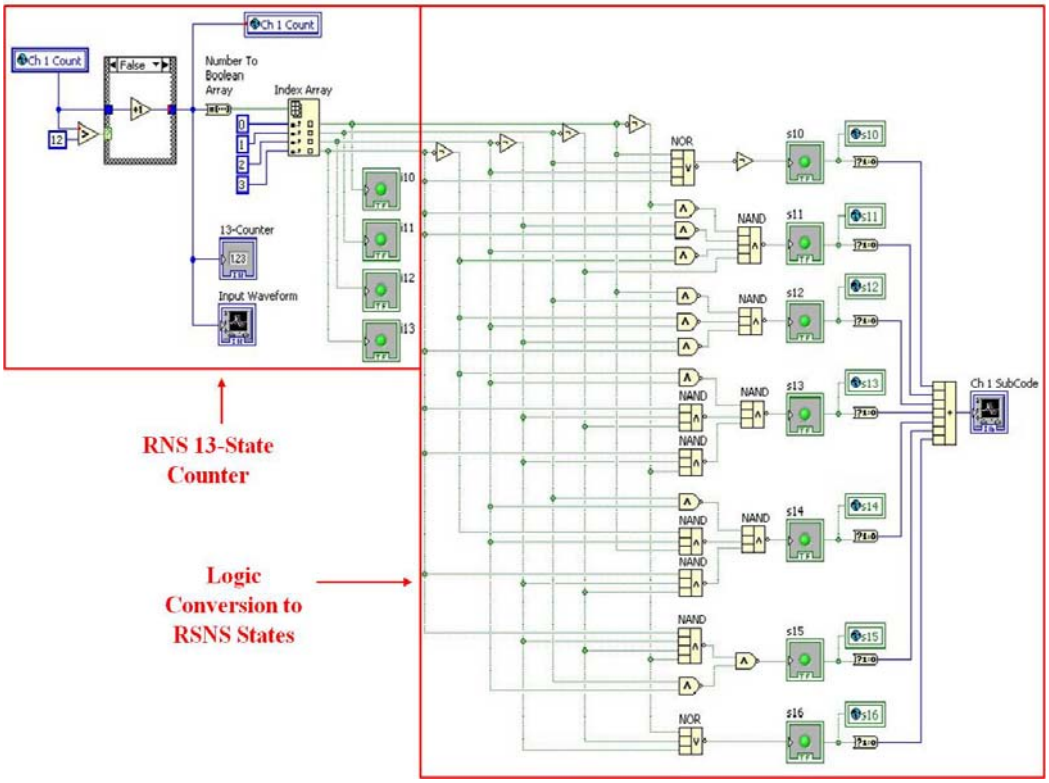

Figure 47. LabVIEW Channel 1 Thermometer Code Generator.

Note that DeMorgan's Theorem was used to achieve a convenient pipelined implementation of all bits in the three channels.

2. Channel 2

The logic table for Channel 2 is shown in Table 4. The RNS states are represented by the bits $\{i_{23}, i_{22}, i_{21}, i_{20}\}$ and are the inputs to the Channel 2 thermometer code generator. The desired thermometer code output is represented by the RSNS bits $\{s_{27}, s_{26}, s_{25}, s_{24}, s_{23}, s_{22}, s_{21}, s_{20}\}$.

Table 4. Channel 2 ($m_2 = 8$) Logic Table.

RNS State	i_{23}	i_{22}	i_{21}	i_{20}	**RSNS State**	s_{27}	s_{26}	s_{25}	s_{24}	s_{23}	s_{22}	s_{21}	s_{20}
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	1	0	0	0	0	0	0	0	1
2	0	0	1	0	2	0	0	0	0	0	0	1	1
3	0	0	1	1	3	0	0	0	0	0	1	1	1
4	0	1	0	0	4	0	0	0	0	1	1	1	1
5	0	1	0	1	5	0	0	0	1	1	1	1	1
6	0	1	1	0	6	0	0	1	1	1	1	1	1
7	0	1	1	1	7	0	1	1	1	1	1	1	1
8	1	0	0	0	8	1	1	1	1	1	1	1	1
9	1	0	0	1	7	0	1	1	1	1	1	1	1
10	1	0	1	0	6	0	0	1	1	1	1	1	1
11	1	0	1	1	5	0	0	0	1	1	1	1	1
12	1	1	0	0	4	0	0	0	0	1	1	1	1
13	1	1	0	1	3	0	0	0	0	0	1	1	1
14	1	1	1	0	2	0	0	0	0	0	0	1	1
15	1	1	1	1	1	0	0	0	0	0	0	0	1

Karnaugh maps are used to carry out logic minimization for Table 4, shown in Figure 48.

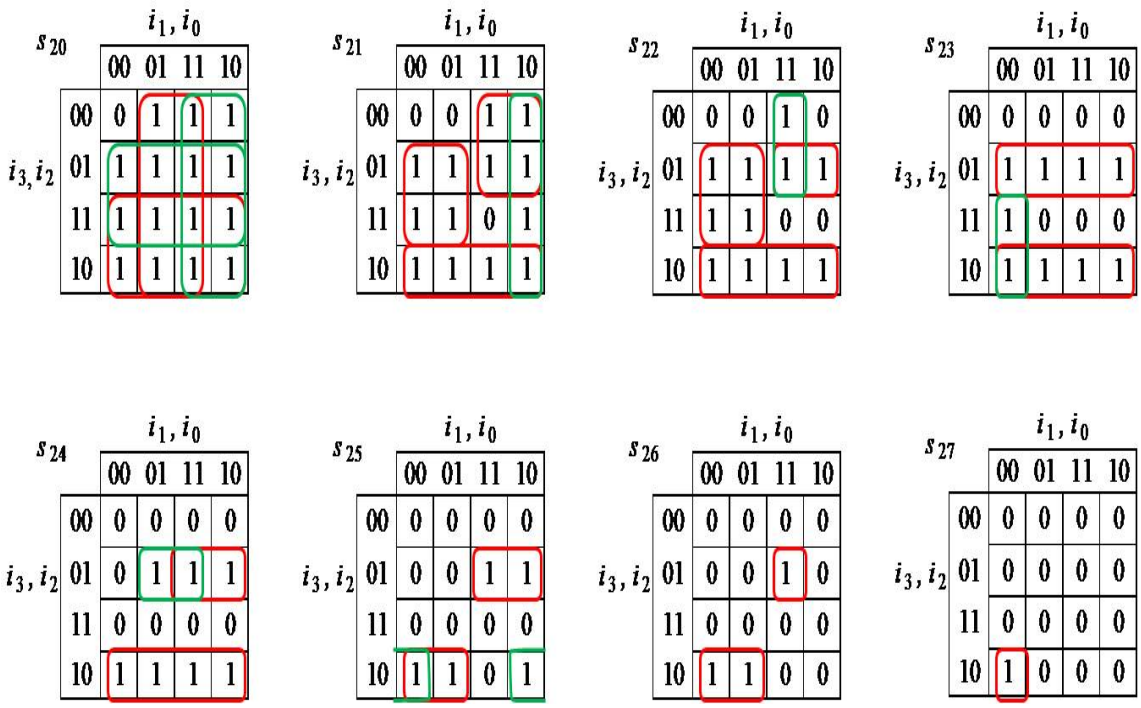

Figure 48. Channel 2 Logic Minimization of Table 4 (After [4]).

The logic minimization in Figure 48 is used to produce the logic equations

$$
\begin{aligned}
s_{20} &= i_3 + i_2 + i_1 + i_0 = \overline{\overline{i_3 + i_2 + i_1 + i_0}} \\
s_{21} &= i_3 i_2 + i_3 \overline{i_1} + \overline{i_2} i_1 + i_1 \overline{i_0} = \overline{\left(\overline{\overline{i_3} i_2}\right)\left(\overline{i_3 \overline{i_1}}\right)\left(\overline{\overline{i_2} i_1}\right)\left(\overline{i_1 \overline{i_0}}\right)} \\
s_{22} &= i_3 \overline{i_2} + i_2 \overline{i_1} + \overline{i_3} i_1 i_0 + \overline{i_3} i_2 i_1 = \overline{\left(\overline{i_3 \overline{i_2}}\right)\left(\overline{i_2 \overline{i_1}}\right)\left(\overline{\overline{i_3} i_1 i_0}\right)\left(\overline{\overline{i_3} i_2 i_1}\right)} \\
s_{23} &= \overline{i_3} i_2 + i_3 \overline{i_2} + i_3 \overline{i_1} \overline{i_0} = \overline{\left(\overline{\overline{i_3} i_2}\right)\left(\overline{i_3 \overline{i_2}}\right)\left(\overline{i_3 \overline{i_1} \overline{i_0}}\right)} \\
s_{24} &= i_3 \overline{i_2} + \overline{i_3} i_2 i_0 + \overline{i_3} i_2 i_1 = \overline{\left(\overline{i_3 \overline{i_2}}\right)\left(\overline{\overline{i_3} i_2 i_0}\right)\left(\overline{\overline{i_3} i_2 i_1}\right)} \\
s_{25} &= \overline{i_3} i_2 i_1 + i_3 \overline{i_2} \overline{i_1} + i_3 \overline{i_2} \overline{i_0} = \overline{\left(\overline{\overline{i_3} i_2 i_1}\right)\left(\overline{i_3 i_2 \overline{i_1}}\right)\left(\overline{i_3 i_2 \overline{i_0}}\right)} \\
s_{26} &= i_3 \overline{i_2} \overline{i_1} + \overline{i_3} i_2 i_1 i_0 = \overline{\left(\overline{i_3 \overline{i_2} \overline{i_1}}\right)\left(\overline{\overline{i_3} i_2 i_1 i_0}\right)} \\
s_{27} &= i_3 \overline{i_2} \overline{i_1} \overline{i_0} = \overline{i_3 + \overline{i_2} + \overline{i_1} + \overline{i_0}}
\end{aligned}
\tag{43}
$$

These logic equations are used to implement the Channel 2 thermometer code generator in LabVIEW, shown in Figure 49.

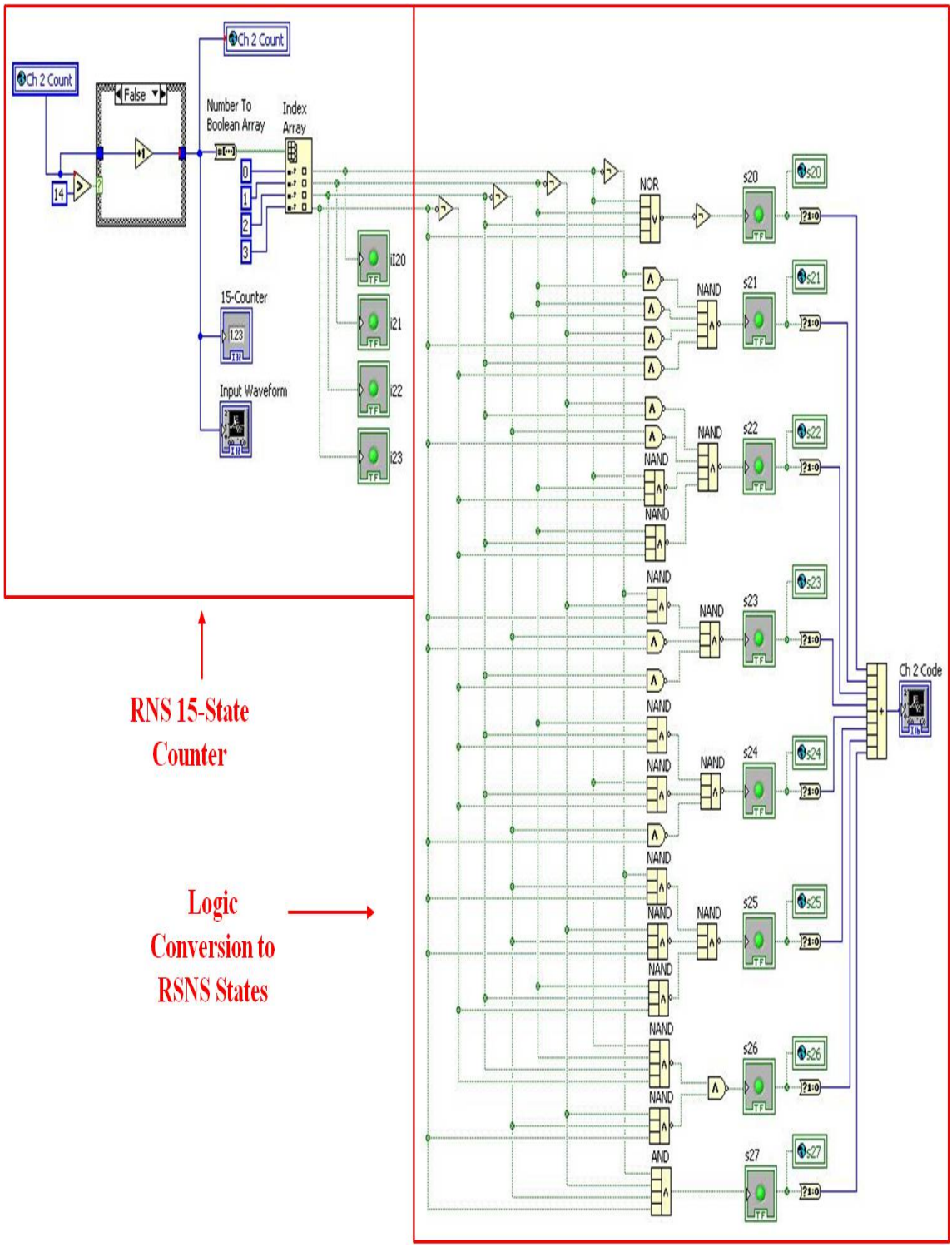

Figure 49. LabVIEW Channel 2 Thermometer Code Generator.

3. Channel 3

The logic table for Channel 3 is shown in Table 5. The RNS states are represented by the bits $\{i_{34}, i_{33}, i_{32}, i_{31}, i_{30}\}$ and are the inputs to the Channel 3 thermometer code generator. The desired thermometer code output is represented by the RSNS bits $\{s_{38}, s_{37}, s_{36}, s_{35}, s_{34}, s_{33}, s_{32}, s_{31}, s_{30}\}$.

Table 5. Channel 3 ($m_3 = 9$) Logic Table.

RNS State	i_{34}	i_{33}	i_{32}	i_{31}	i_{30}	**RSNS State**	s_{38}	s_{37}	s_{36}	s_{35}	s_{34}	s_{33}	s_{32}	s_{31}	s_{30}
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1
2	0	0	0	1	0	2	0	0	0	0	0	0	0	1	1
3	0	0	0	1	1	3	0	0	0	0	0	0	1	1	1
4	0	0	1	0	0	4	0	0	0	0	0	1	1	1	1
5	0	0	1	0	1	5	0	0	0	0	1	1	1	1	1
6	0	0	1	1	0	6	0	0	0	1	1	1	1	1	1
7	0	0	1	1	1	7	0	0	1	1	1	1	1	1	1
8	0	1	0	0	0	8	0	1	1	1	1	1	1	1	1
9	0	1	0	0	1	9	1	1	1	1	1	1	1	1	1
10	0	1	0	1	0	8	0	1	1	1	1	1	1	1	1
11	0	1	0	1	1	7	0	0	1	1	1	1	1	1	1
12	0	1	1	0	0	6	0	0	0	1	1	1	1	1	1
13	0	1	1	0	1	5	0	0	0	0	1	1	1	1	1
14	0	1	1	1	0	4	0	0	0	0	0	1	1	1	1
15	0	1	1	1	1	3	0	0	0	0	0	0	1	1	1
16	1	0	0	0	0	2	0	0	0	0	0	0	0	1	1
17	1	0	0	0	1	1	0	0	0	0	0	0	0	0	1

Karnaugh maps are used to carry out logic minimization for Table 5, shown in Figure 50.

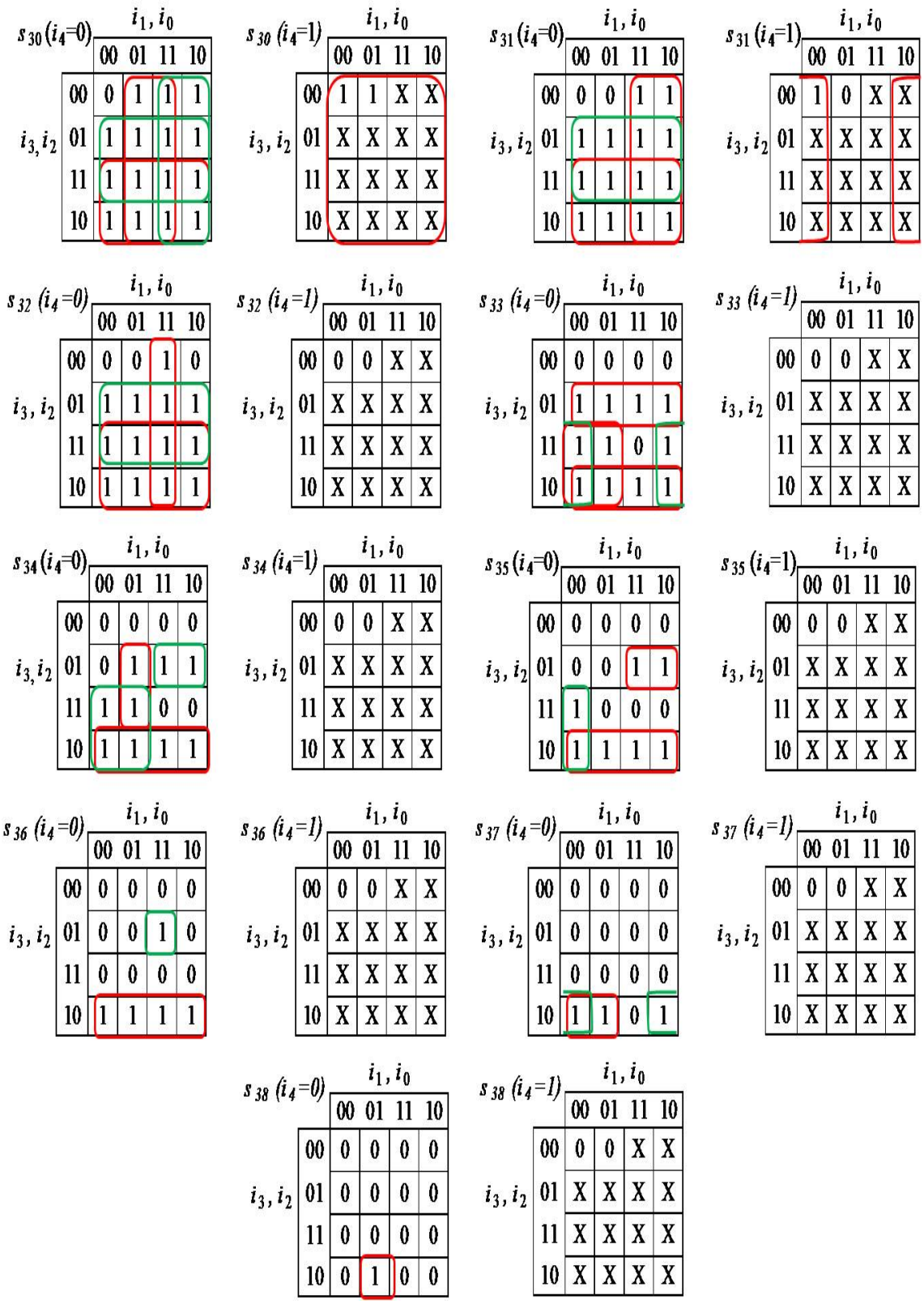

Figure 50. Channel 3 Logic Minimization of Table 5 (After [4]).

The logic minimization in Figure 50 is used to produce the logic equations

$$
\begin{aligned}
s_{30} &= i_4 + i_3 + i_2 + i_1 + i_0 = \overline{\left(\overline{i_4 + i_3 + i_2}\right)\left(\overline{i_1 + i_0}\right)} \\
s_{31} &= i_4\overline{i_0} + i_3 + i_2 + i_1 = \overline{\left(\overline{i_4\overline{i_0}}\right)\left(\overline{i_3 + i_2 + i_1}\right)} \\
s_{32} &= i_3 + i_2 + i_1 i_0 = \overline{\left(\overline{i_3 + i_2}\right)\left(\overline{i_1 i_0}\right)} \\
s_{33} &= \overline{i_3} i_2 + i_3\overline{i_1} + i_3\overline{i_2} + i_3\overline{i_0} = \overline{\left(\overline{\overline{i_3} i_2}\right)\left(\overline{i_3\overline{i_1}}\right)\left(\overline{i_3\overline{i_2}}\right)\left(\overline{i_3\overline{i_0}}\right)} \\
s_{34} &= i_3\overline{i_2} + i_3\overline{i_1} + i_2\overline{i_1} i_0 + \overline{i_3} i_2 i_1 = \overline{\left(\overline{i_3\overline{i_2}}\right)\left(\overline{i_3\overline{i_1}}\right)\left(\overline{i_2\overline{i_1} i_0}\right)\left(\overline{\overline{i_3} i_2 i_1}\right)} \\
s_{35} &= i_3\overline{i_2} + i_3\overline{i_1}\,\overline{i_0} + \overline{i_3} i_2 i_1 = \overline{\left(\overline{i_3\overline{i_2}}\right)\left(\overline{i_3\overline{i_1}\,\overline{i_0}}\right)\left(\overline{\overline{i_3} i_2 i_1}\right)} \\
s_{36} &= \overline{i_3} i_2 i_1 i_0 + i_3\overline{i_2} = \overline{\left(\overline{\overline{i_3} i_2 i_1 i_0}\right)\left(\overline{i_3\overline{i_2}}\right)} \\
s_{37} &= i_3\overline{i_2}\,\overline{i_1} + i_3\overline{i_2}\,\overline{i_0} = \overline{\left(\overline{i_3\overline{i_2}\,\overline{i_1}}\right)\left(\overline{i_3\overline{i_2}\,\overline{i_0}}\right)} \\
s_{38} &= i_3\overline{i_2}\,\overline{i_1} i_0 = \overline{i_3 + \overline{i_2} + \overline{i_1} + i_0}
\end{aligned}
\tag{44}
$$

These logic equations are used to implement the Channel 3 thermometer code generator in LabVIEW, shown in Figure 51.

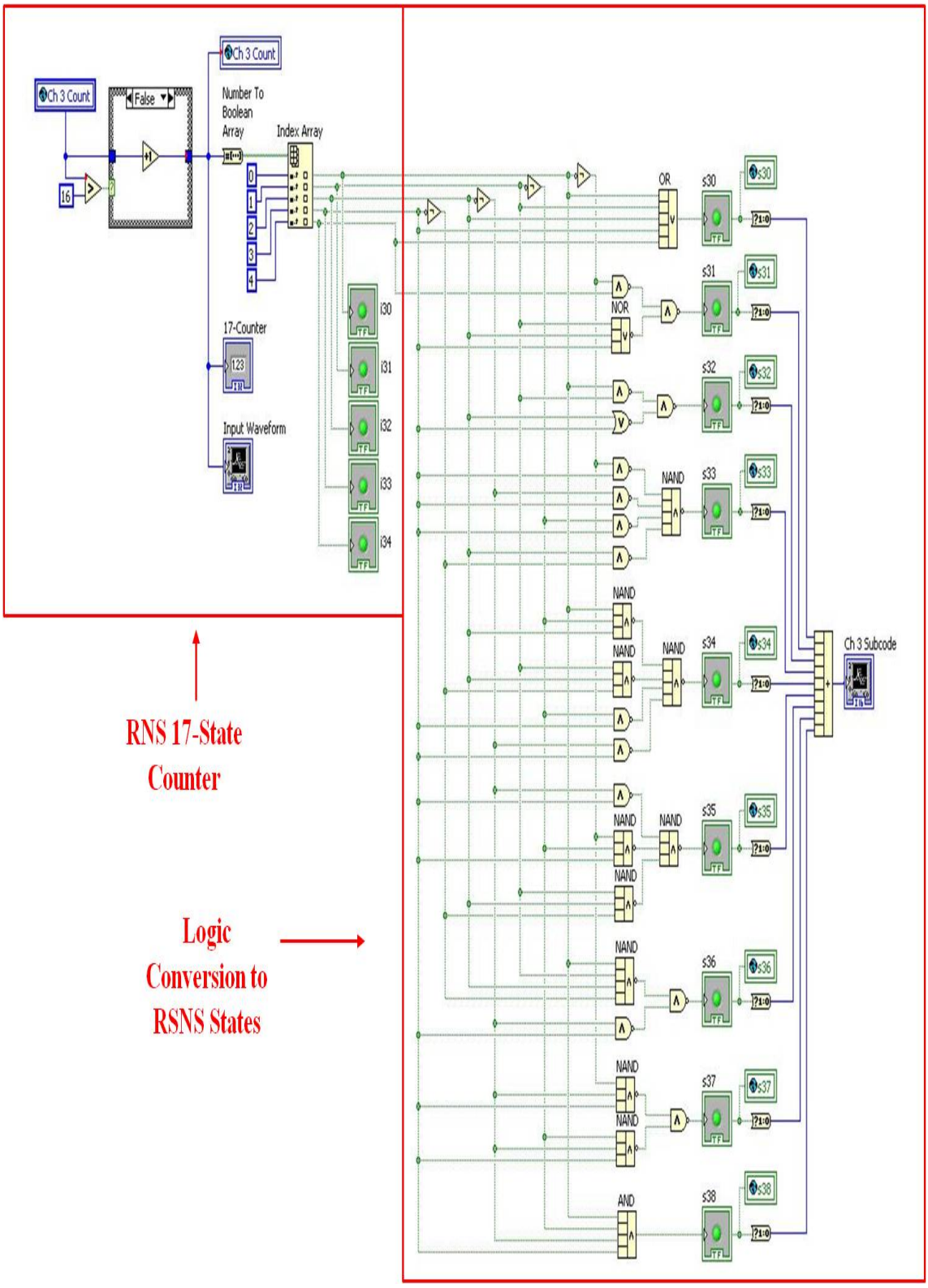

Figure 51. LabVIEW Channel 3 Thermometer Code Generator.

B. SIMULATION MODEL

To test the RSNS-to-binary conversion, a simulation model was created with the thermometer code generators from the three channels enclosed within a case structure, shown in Figure 52. A 'Master Clock' global variable was created to determine the starting channel and to step through each of the three channels. Three 'Channel Count' global variables were used to set the channel start positions.

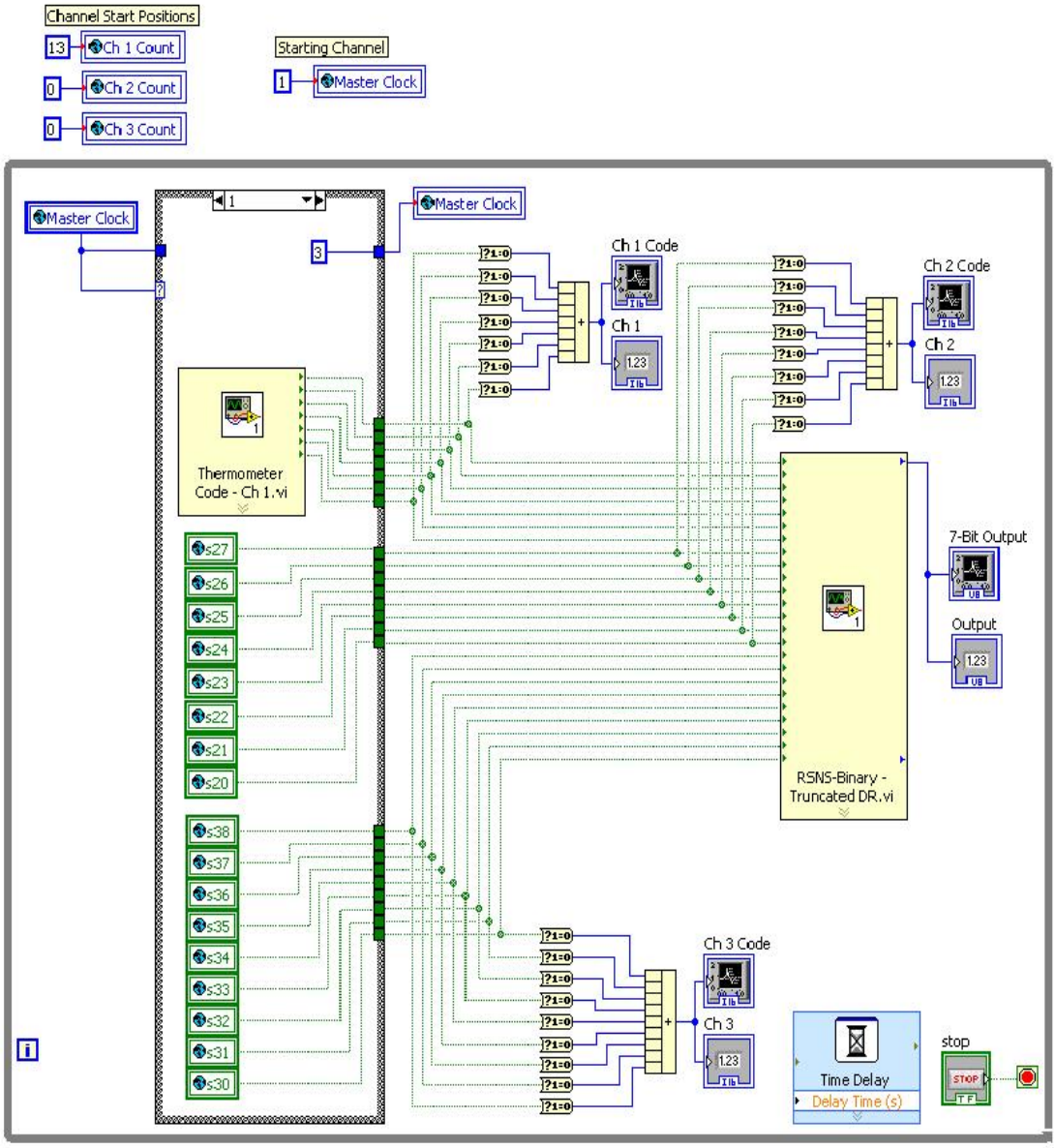

Figure 52. LabVIEW Simulation of RSNS-to-Binary Conversion.

C. SIMULATION RESULTS

After running the simulation model in Figure 52, the thermometer code inputs and the RSNS-to-binary conversion output can be viewed on the respective scopes located in the LabVIEW Front Panel.

1. Truncated Dynamic Range

The thermometer code inputs and the simulated RSNS-to-binary output with no ambiguities within the truncated DR of 126 are shown in Figure 53. The RSNS position index *h* is from 733 to 857.

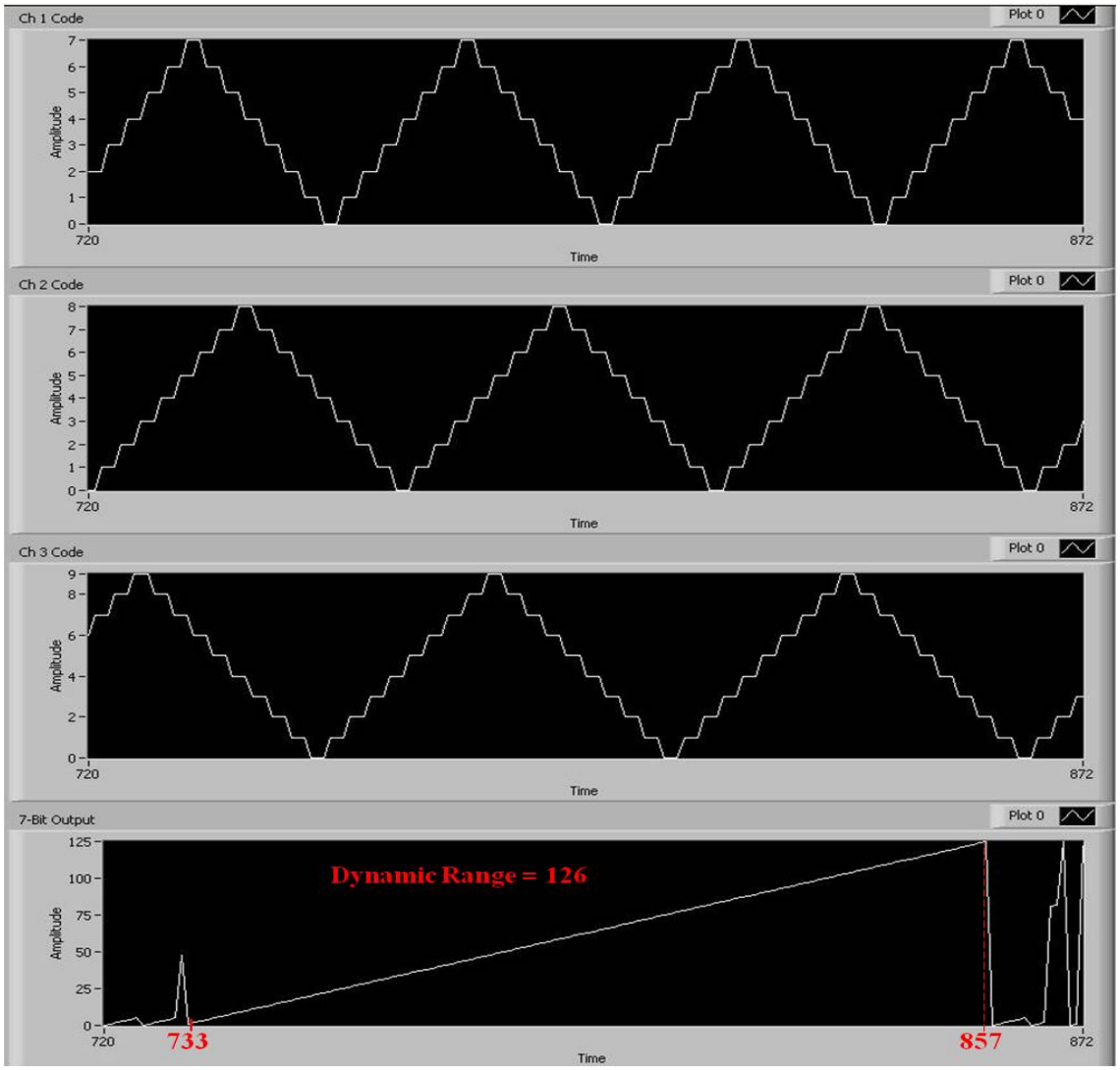

Figure 53. Simulated RSNS-to-Binary Output (Truncated DR Case).

2. Full Dynamic Range

To achieve the full DR, the encoder circuit in Figure 44 was used in place of Figure 43 in the RSNS-to-binary conversion. The ambiguities in the simulated RSNS-to-binary output within the full DR of 133 are shown in Figure 54. The RSNS position index *h* is from 733 to 865.

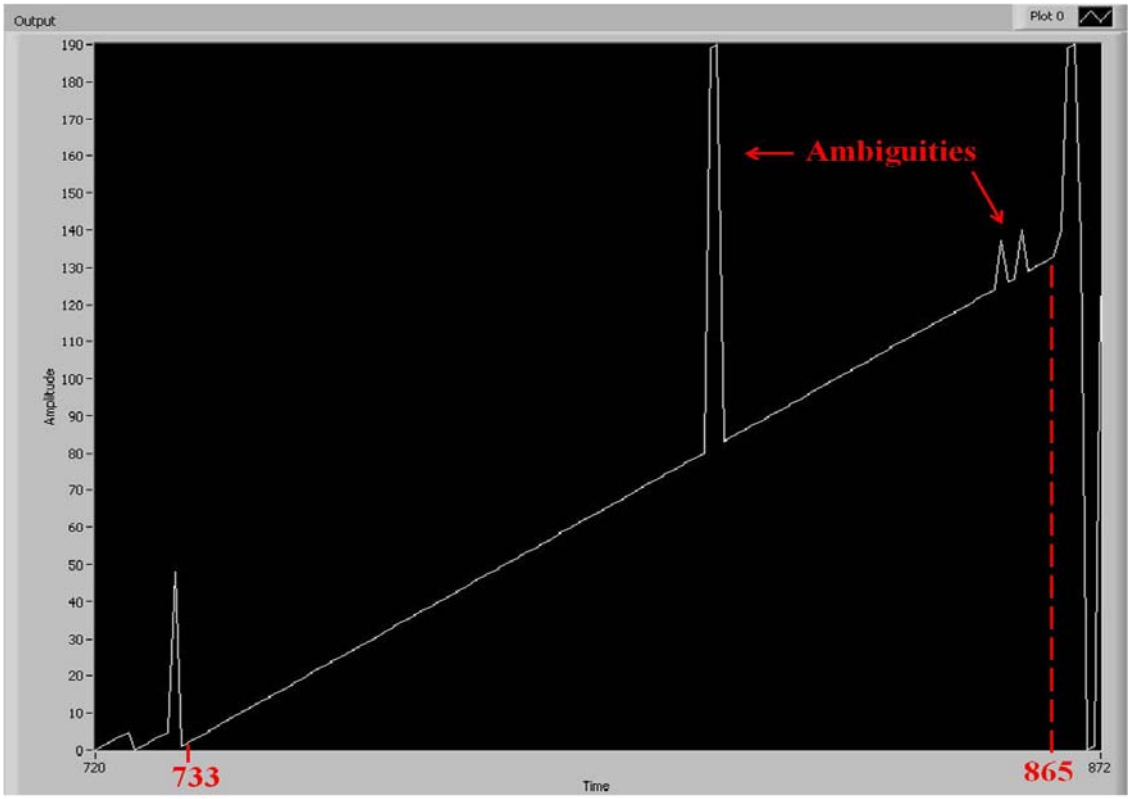

Figure 54. Simulated RSNS-to-Binary Output (Full DR Case with Ambiguities).

3. LPS Priority Circuit

The ambiguities in Figure 54 occur because the DR does not start with a Sub-Channel 0 vector and end with a Sub-Channel 2 vector, as required in Figure 9. As a result, a start position with a Sub-Channel 0 vector $(h_1 = 732)$ has to be chosen outside the DR, during the DR compression process in Chapter II.

As the DR is not truncated to a length evenly divisible by six, there may be ambiguities where two LPS equations will be asserted at the same time – one of the boundary LPS equations (i.e. *LPS0 or LPS22*) and *LPSX*. If the full DR is to be maintained with no ambiguities, the boundary LPS equation should always be ignored in favor of the *LPSX* solution.

This can be done by implementing an encoder with an addition of a LPS Priority circuit to check for such cases, and giving priority to the center 21 NAND gates over the first and last NAND gates. This is shown in Figure 55.

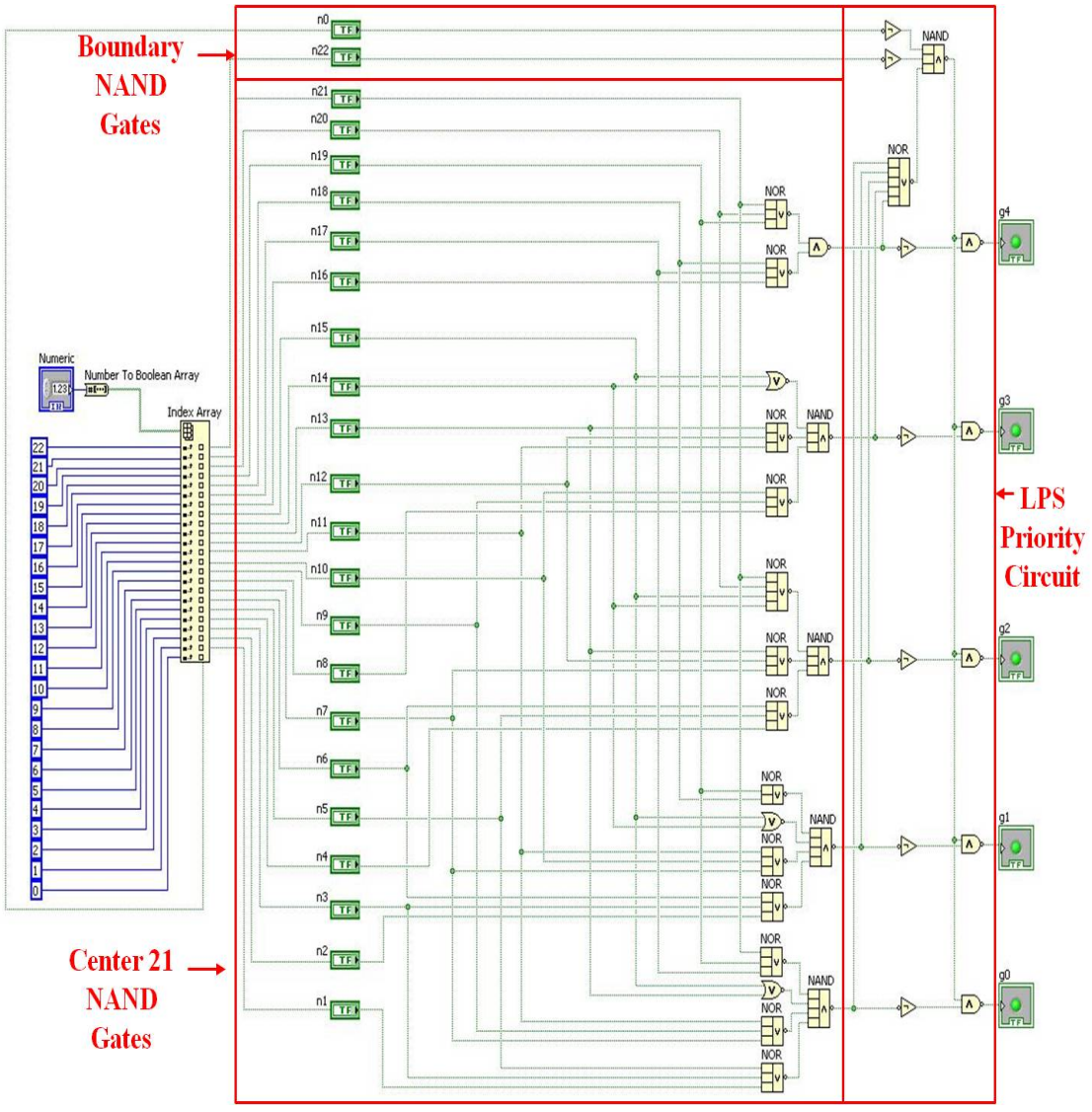

Figure 55. LabVIEW Schematics of Encoder with LPS Priority Circuit.

4. Full Dynamic Range with LPS Priority Circuit

To achieve the full DR with no ambiguities, the encoder circuit in Figure 55 was used in place of Figures 43 and 44 in the RSNS-to-binary conversion. From Figure 56, it can be seen that there are no ambiguities in the simulated RSNS-to-binary output within the full DR of 133. The RSNS position index *h* is from 733 to 865.

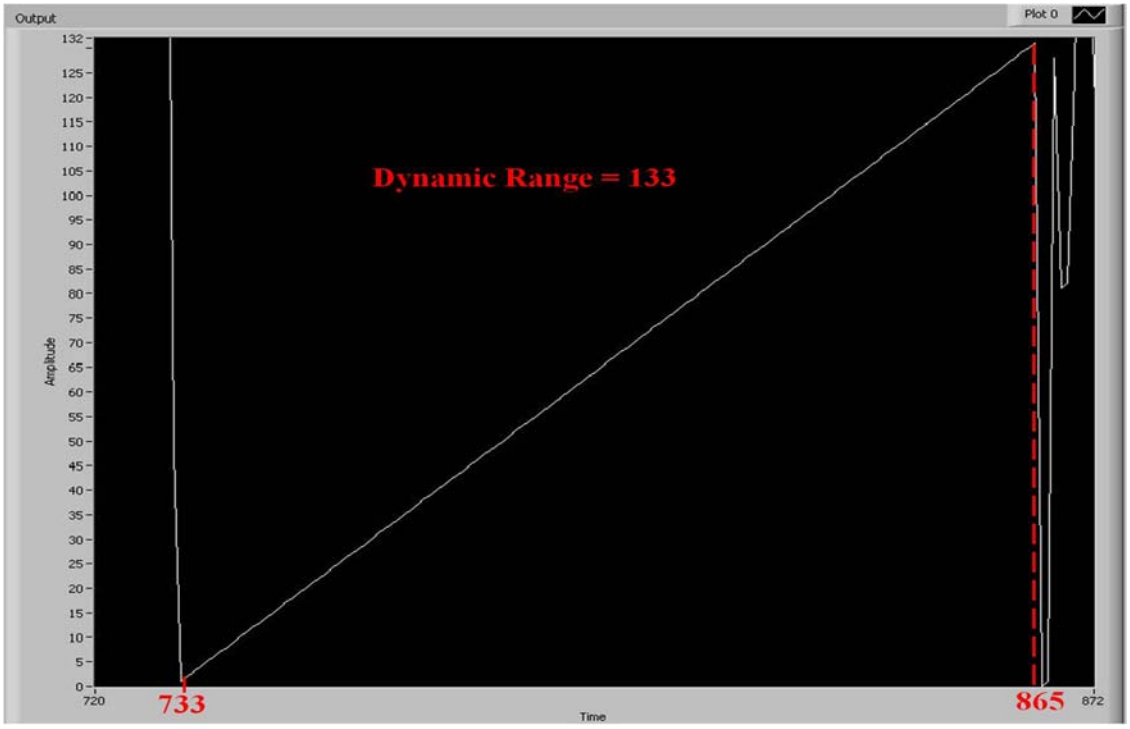

Figure 56. Simulated RSNS-to-Binary Output (Full DR Case with No Ambiguities).

Table 6. Comparison of Dynamic Range.

Model	Position Index *h*	Dynamic Range	Ambiguities?
Truncated DR	733 – 857	126	No
Full DR	733 – 865	133	Yes
Full DR with LPS Priority	733 – 865	133	No

A comparison of the dynamic range achieved by the three different cases is shown in Table 6. The simulation proves that an eight-bit DR of 133 can be achieved for a three-channel RSNS ADC with Moduli $m_1 = 7$, $m_2 = 8$ and $m_3 = 9$. This is in agreement with the theoretical DR calculated from (5), and a one-bit improvement over that achieved in [4].

Simulation of the RSNS-to-binary conversion algorithm was successfully carried out to verify that it is working properly in this chapter. In the next chapter, the implementation of the comparator circuit and RSNS-to-binary conversion algorithm on a FPGA to allow the code to run at a higher sampling rate and frequency is elaborated. The FPGA is then integrated with the front-end photonics implementation to form the overall folding ADC architecture.

VI. ADC INTEGRATION

The integration process to form an overall folding ADC system is documented in this chapter. The first section of this chapter contains an illustration of the implementation of the comparator circuit design and RSNS-to-binary conversion algorithm on an NI FPGA to form the DDS module. The integration of the DDS module with the front-end PES module in [5] is highlighted in the second section.

A. DIGITAL DECODING SUB-SYSTEM

An overview of the DDS module from the bank of comparators to the FPGA is shown in Figure 57. An NI-9215 Analog Input Module is used to connect the three analog photo-detector inputs from the front-end PES module to the NI-9111 Chassis with an onboard XILINX Virtex-5 LX30 FPGA. The FPGA output is then sent to the NI-9012 CompactRIO (cRIO) Real-Time Controller (RTC), and can be saved to a file using a LabVIEW Host Interface VI.

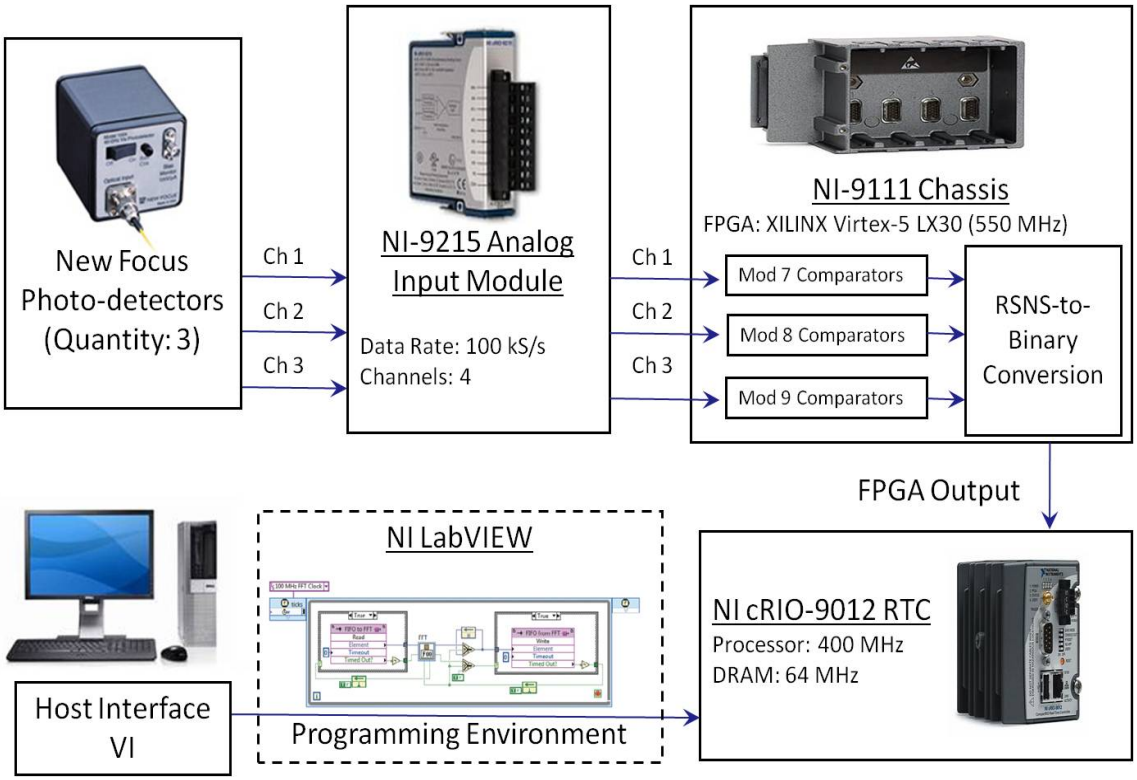

Figure 57. DDS Module Setup.

The comparator speed was a limiting factor in the ADC system speed as actual comparator ICs were used for sampling outside the FPGA in [3]. The key improvement made in the DDS module setup in Figure 57 is that the comparator circuits reside in the FPGA, allowing them to sample at a rate equal to the FPGA speed.

Lastly, the comparator circuit and RSNS-to-binary conversion logics are run on the FPGA to guarantee a higher FPGA execution speed as opposed to running it on the RTC with a lower processing speed.

B. COMPARATORS

In the actual ADC architecture, comparators are used to convert the analog photo-detector outputs into thermometer codes for each channel in place of thermometer code generators. The threshold values for the comparators can be expressed as [3]:

$$T(k,m_i) = V_{fs_i} * \cos^2\left[\frac{\pi}{2} - \left(\frac{\pi(kN+1)}{P_{RSNS}} + \frac{\pi}{2P_{RSNS}}\right)\right] \tag{45}$$

where k = 0, 1, …, $m_\text{i} - 1$, V_fs = full-scale voltage for channel Moduli m_i, N = number of channels, and P_RSNS = period of RSNS channel = $2Nm_i$.

The full-scale voltage is defined as the maximum amplitude of the modulated signal for each channel and determined in [5].

1. Channel 1

Using (45) with $V_{fs} = 2.5$ (from [5]), we calculate the quantized threshold values for Channel 1 as:

$$T(k,m_1) = [0.0313,\ 0.2727,\ 0.7076,\ 1.2500,\ 1.7924,\ 2.2273,\ 2.4687]. \tag{46}$$

The implementation of the Channel 1 comparators in LabVIEW using the threshold voltages calculated in (46) is shown in Figure 58.

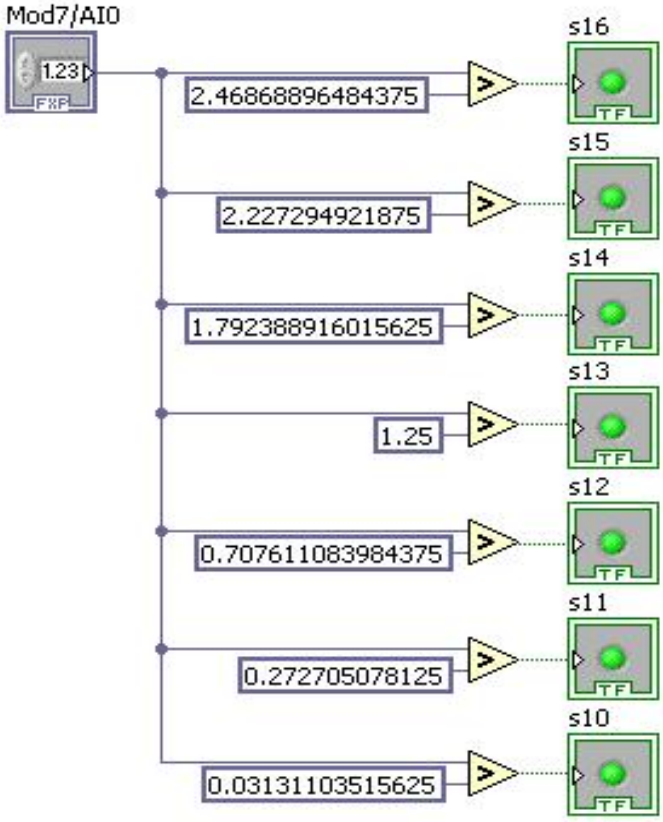

Figure 58. LabVIEW Schematics of Channel 1 Comparators.

2. Channel 2

Using (45) with $V_{fs} = 5.11$ (provided from [5]), we calculate the quantized threshold values for Channel 2 as

$$T(k, m_2) = [0.0419,\ 0.4306,\ 1.1355, 2.0565, 3.0535, 3.9745, 4.6794, 5.0609]. \quad (47)$$

The implementation of the Channel 2 comparators in LabVIEW using the threshold voltages calculated in (47) is shown in Figure 59.

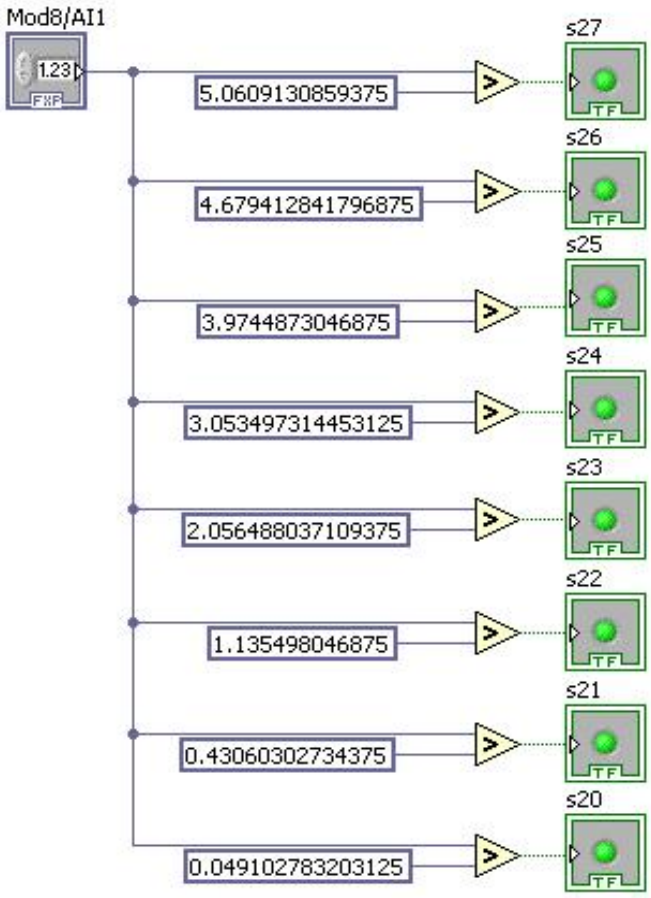

Figure 59. LabVIEW Schematics of Channel 2 Comparators.

3. Channel 3

Using (45) with $V_{fs} = 4.96$ (provided from [5]), we calculate the quantized threshold values for Channel 3 as

$$T(k, m_3) = [0.0377,\ 0.3323,\ 0.8859,\ 1.6318,\ 2.4800,\ 3.3282,\ 4.0741,\ 4.6277,\ 4.9223]. \quad (48)$$

The implementation of the Channel 3 comparators in LabVIEW using the threshold voltages calculated in (48) is shown in Figure 60.

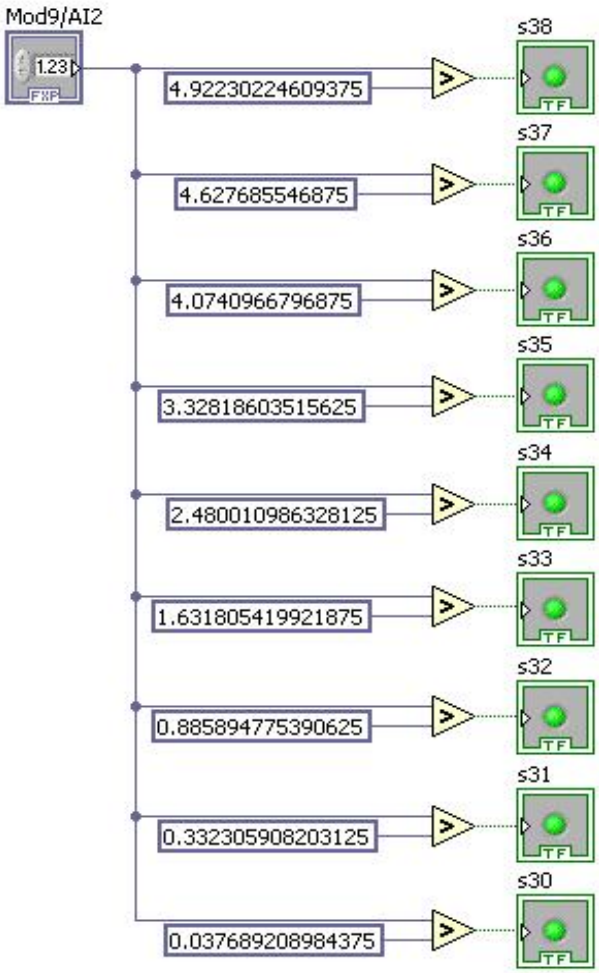

Figure 60. LabVIEW Schematics of Channel 3 Comparators.

C. FPGA IMPLEMENTATION

In order to implement the comparators and RSNS-to-binary conversion on the FPGA, a LabVIEW FPGA project was created as 'FPGA789.lvproj'. An overview of this FPGA project is shown in Figure 61.

The FPGA has a Dynamic Memory Allocation (DMA) First-In-First-Out (FIFO) buffer, which is used to transfer data from the FPGA to a host computer for this project. LabVIEW also requires the creation of an FPGA VI to run the necessary codes on the FPGA and a Host VI to communicate with the FPGA VI, which will be explained in the next two sections.

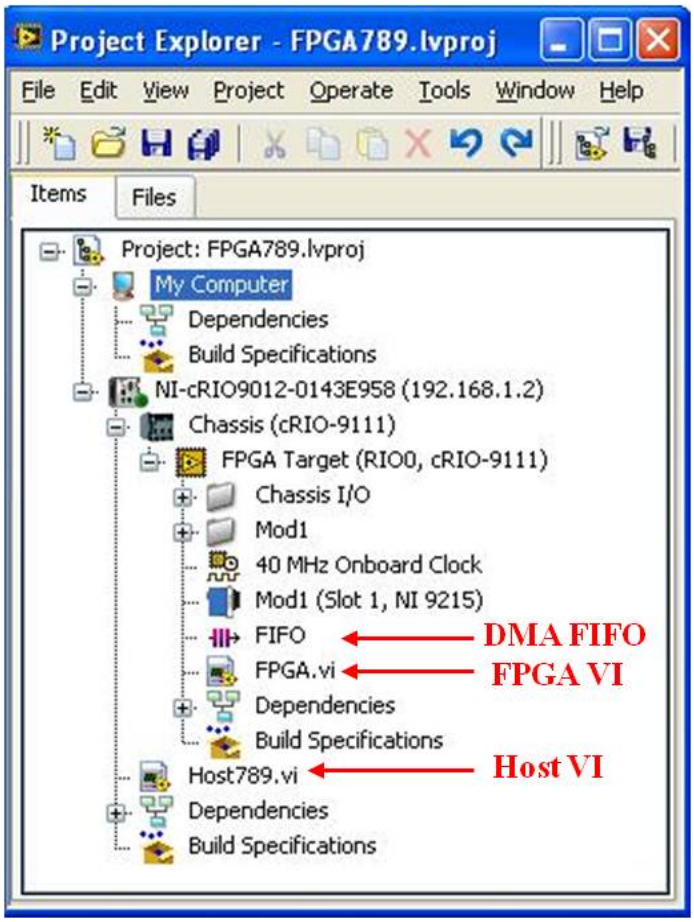

Figure 61. LabVIEW FPGA Project Overview.

1. FPGA VI

In the FPGA VI, the three photo-detector inputs were read in using the NI-9215 Analog Input Module, sampled using the comparator circuits, and processed using the RSNS-to-binary conversion algorithm. The RSNS-to-binary output was then stored in the DMA FIFO. The number of elements in the FIFO can be set in Figure 61 by double-clicking on the FIFO icon and changing its properties. The LabVIEW schematics of the FPGA VI are shown in Figure 62.

A point to note is that the thermometer code displays were removed from the Front Panel of the VI, as it slowed down the FPGA execution speed when it was required to display any data.

The truncated DR case was used for the RSNS-to-binary conversion for this integration as the interferometers used in [5] could not create enough folds of the modulated signals to exploit the whole DR.

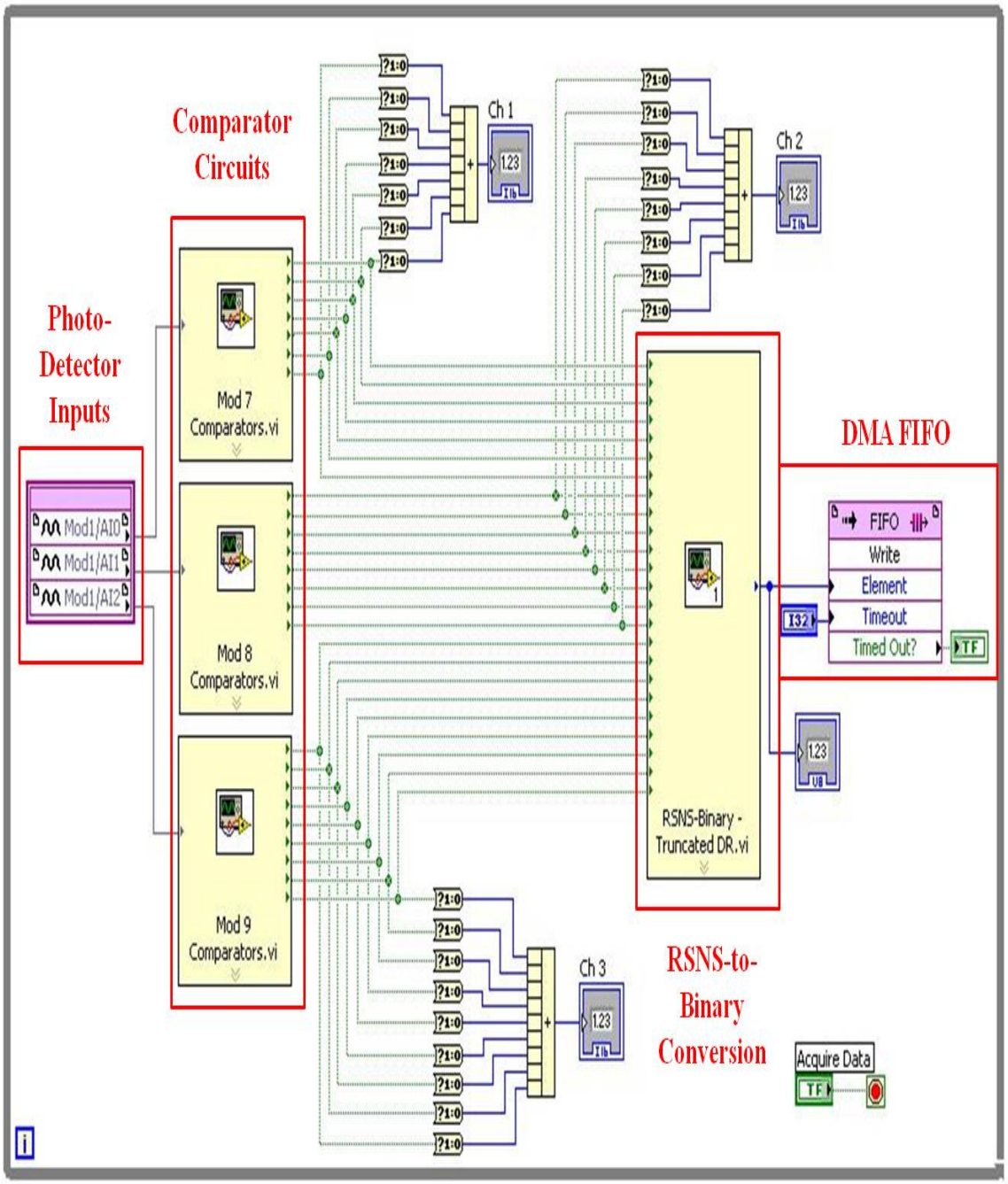

Figure 62. LabVIEW Schematics of FPGA VI.

2. Host Interface VI

In the Host VI, LabVIEW requires that a reference to the FPGA VI be opened before the Host VI can communicate with the FPGA VI. Thereafter, a 'FIFO Read' action was invoked by the Host VI to read the DMA FIFO data in the FPGA VI. The data was then saved in a text file, using the 'Write to Measurement File' VI. The LabVIEW schematics of the Host VI are shown in Figure 63.

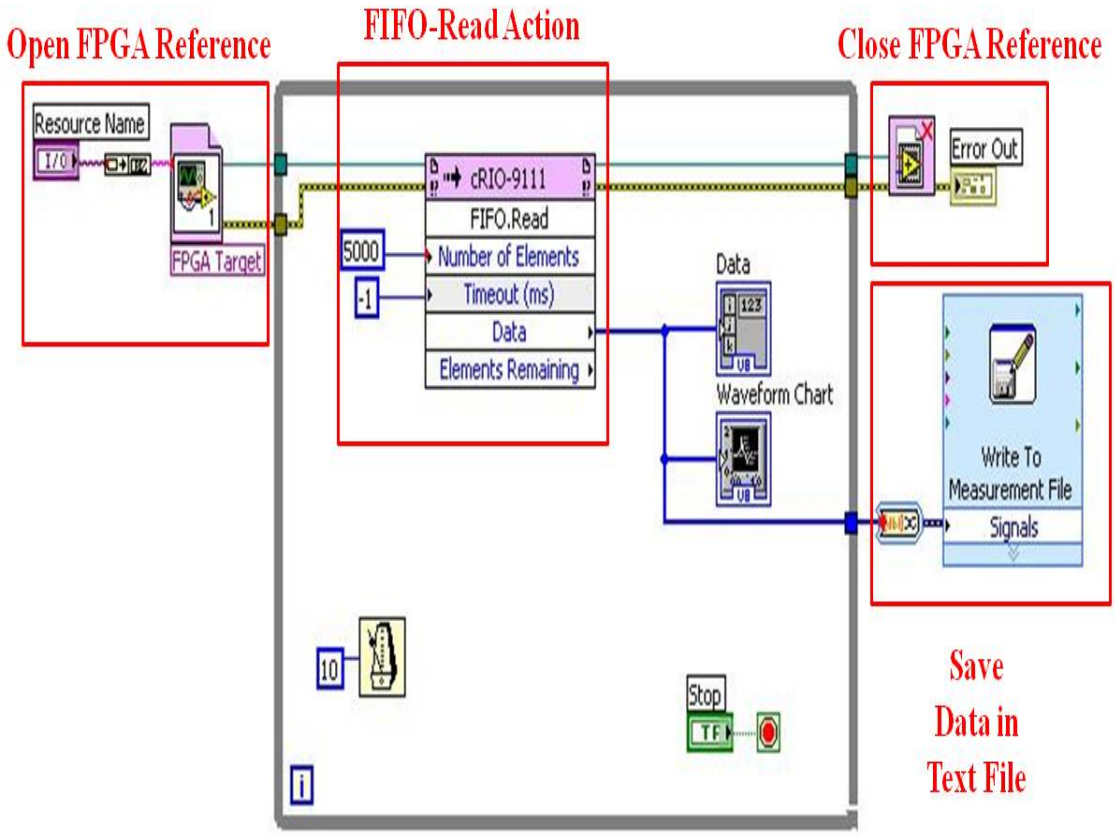

Figure 63. LabVIEW Schematics of Host VI.

3. Results

After compiling and running the FPGA and Host VIs, proper alignment of the modulated signals had to be carried out to ensure that the RSNS vectors were lined up correctly to achieve accurate decoding. The DDS outputs for a 1-kHz triangular and 1-kHz sine input signal are shown in Figures 64 and 65, respectively.

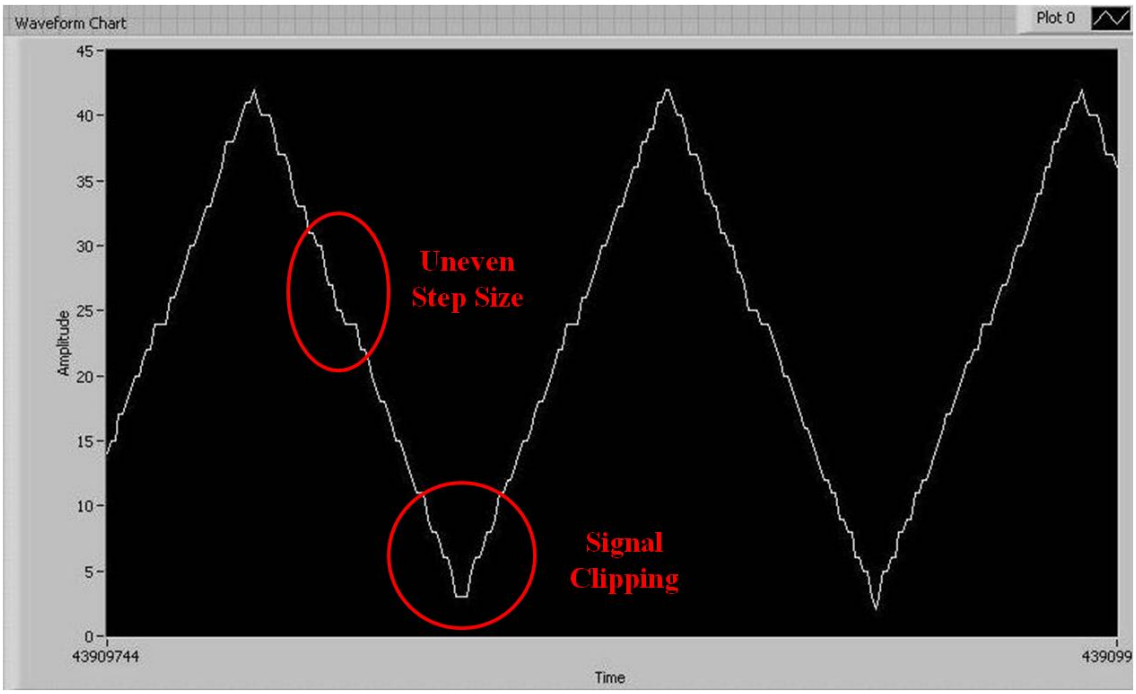

Figure 64. DDS Output for a 1-kHz Triangular Input Signal.

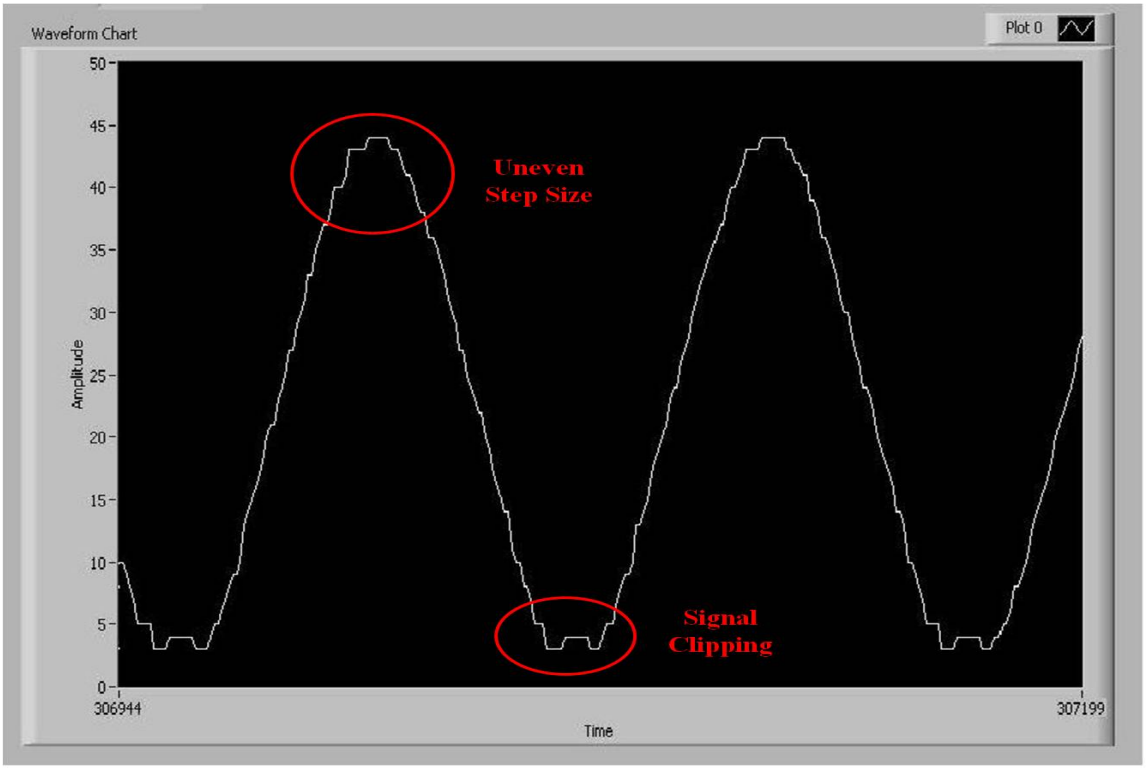

Figure 65. DDS Output for a 1-kHz Sine Input Signal.

It is seen from Figures 64 and 65 that the DDS module was able to decode the input signals with some signal clipping and uneven step sizes in the output waveforms.

Implementations of the comparator circuit and RSNS-to-binary conversion algorithm on the FPGA to form the DDS module and DDS module integration with the front-end PES module to form the overall folding ADC architecture were carried out in this chapter. An analysis of the ADC performance characteristics is provided in the next chapter.

VII. ADC PERFORMANCE

In this chapter, a number of important parameters that describe an ADC's performance are presented. Differential and integral linearity errors are plotted for the ADC to analyze the linearity errors. The ADC dynamic range is determined using a full-scale sinusoid and a Fourier spectrum analysis of the noise floor. Finally, the ADC dynamic performance is characterized by the Signal-to-Noise Ratio (SNR), SNR plus distortion (SINAD), Total Harmonic Distortion (THD) and the Effective Number of Bits (ENOB) parameters.

A. LINEARITY ERRORS

A comparison of the ADC 1-kHz sine input signal and the DDS output signal is shown in Figure 66, where it can be seen that the DDS output signal is able to follow the input signal, with some quantization and linearity errors.

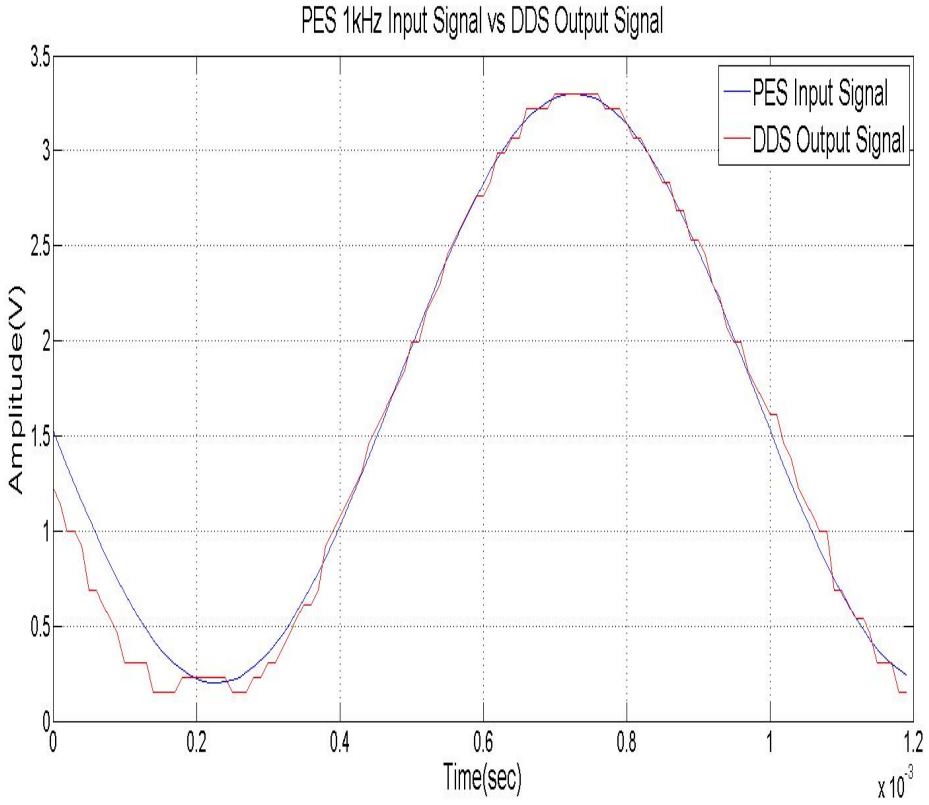

Figure 66. Comparison of PES Input Signal and DDS Output Signal.

The characteristic transfer function of a 1-kHz triangular waveform is used to quantify the linearity errors. A plot of the ADC transfer function using a 1-kHz triangular input signal is shown in Figure 67. The quantization errors of the ADC transfer function in Figure 67 are shown in Figure 68. Quantization error is present as there is no one-to-one correspondence between the input and output voltage.

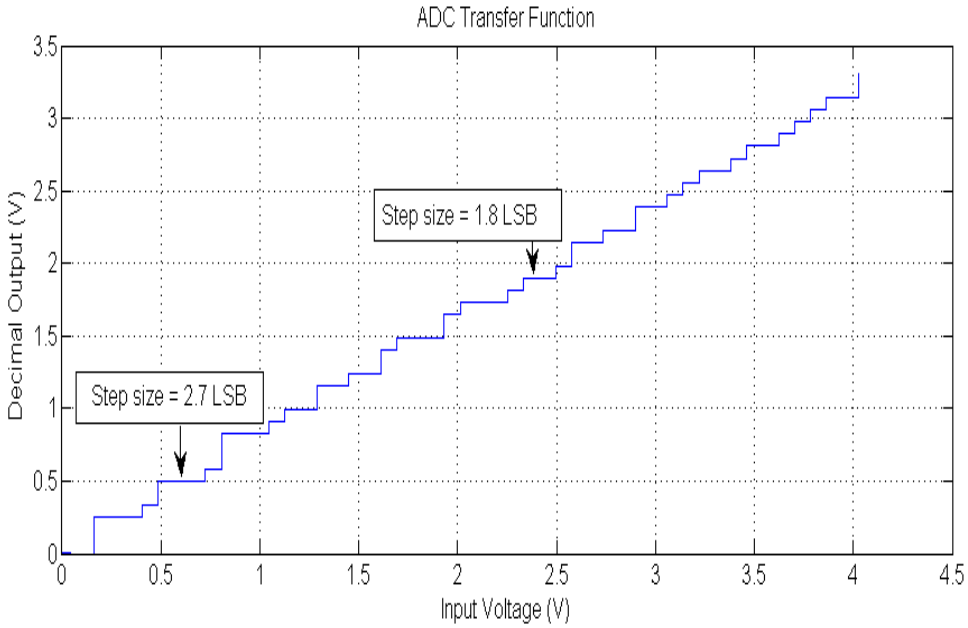

Figure 67. Photonic ADC Transfer Function using a 1-kHz Triangular Input Signal.

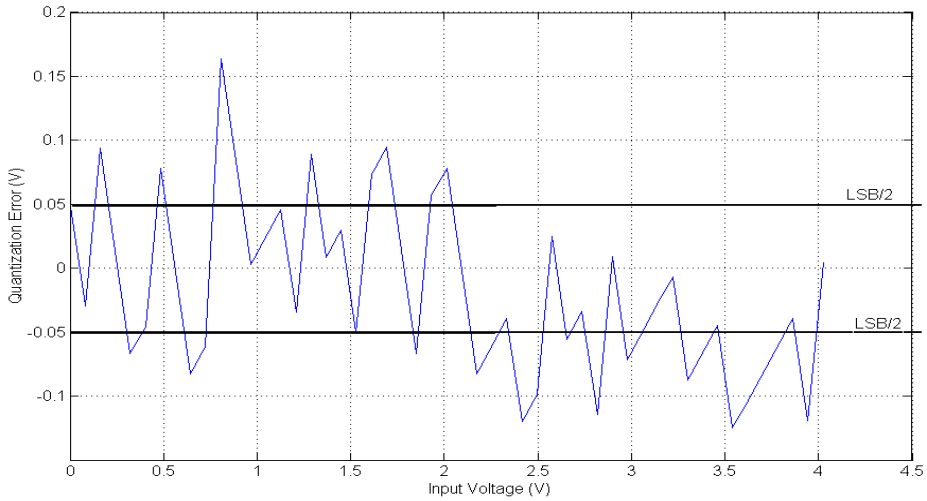

Figure 68. Quantization Errors.

1. ADC Resolution

The LSB size for this RSNS ADC is defined as:

$$LSB_{RSNS} = \frac{V_{fs}}{\hat{M}} \tag{49}$$

where LSB_{RSNS} is the ADC resolution, V_{fs} is the full-scale voltage and $\hat{M}$ is the maximum system dynamic range.

2. Differential Non-Linearity

The Differential Non-Linearity (DNL) is expressed as [10]:

$$DNL_k = \left(V_k - V_{k-1}\right) - LSB \tag{50}$$

where V_k and V_{k-1} are two consecutive code transition points, and $\left(V_k - V_{k-1}\right)$ is the step-size. The DNL is the maximum deviation in the output step-size from the ideal value of one LSB.

3. Integral Non-Linearity

The Integral Non-Linearity (INL) is defined as [10]:

$$INL_j = \sum_{k=1}^{j} DNL_k = V_j - jLSB \tag{51}$$

where $V_j = \sum_{k=1}^{j}\left(V_k - V_{k-1}\right)$ is the sum of the step-size from zero to the j^{th} transition point, and $jLSB$ is the ideal value at that transition point. The INL is the maximum deviation of the input/output characteristic from a straight line passed through its end points. A good ADC typically has linearity error $\leq$ 0.5 LSB. [10]

The linearity errors (step-size, DNL and INL) of the quantized signal in Figure 67 are shown in Figure 69. The step-size plot depicts the length of input voltage

corresponding to each quantization level as the input voltage increases. It shows that the maximum step-size is 2.7 LSB, which corresponds to the code transition points with a maximum DNL value of 1.7 LSB.

A maximum INL value of 7.8 LSB is also obtained in Figure 69. The INL increases with a larger input voltage, indicating that the slope of the ADC transfer function is deviating from that of an ideal ADC transfer function at larger input voltages.

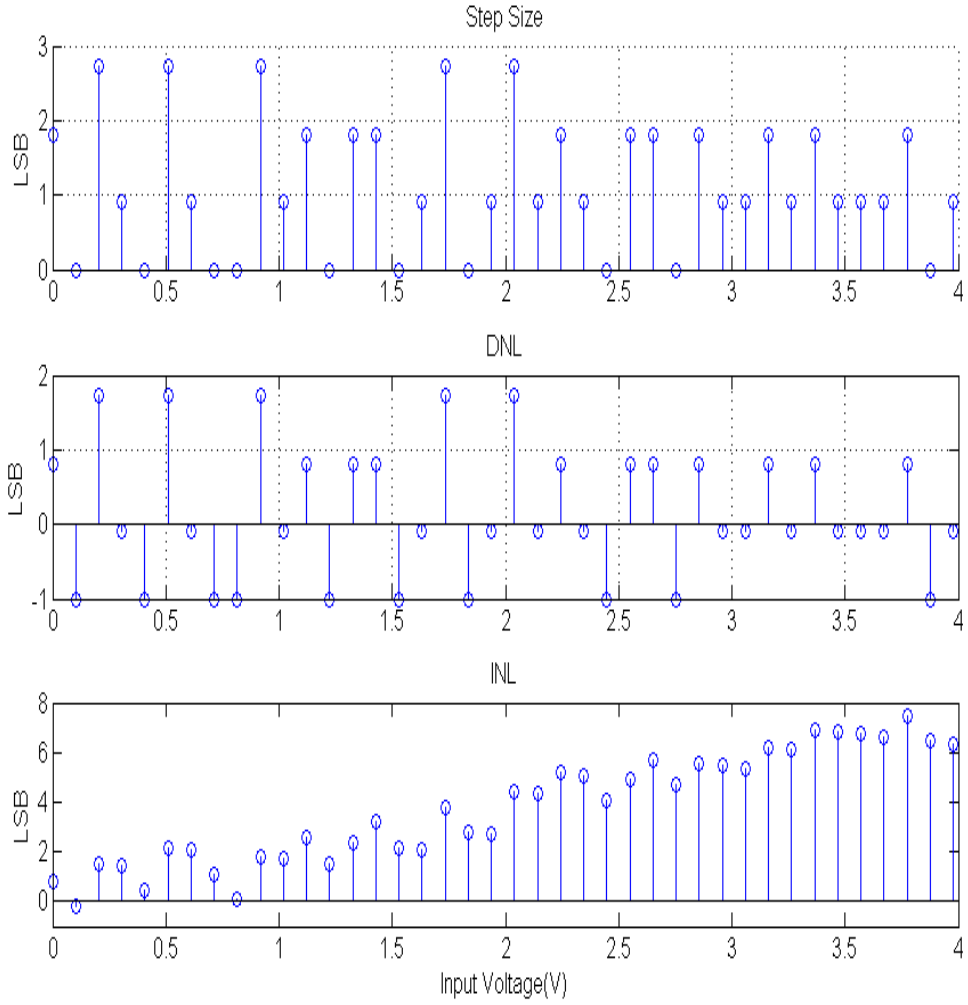

Figure 69. Linearity Parameters – Step-Size, DNL and INL.

B. NOISE FLOOR ANALYSIS

The process for computing the frequency spectrum and analyzing the ADC noise floor is illustrated in Figure 70.

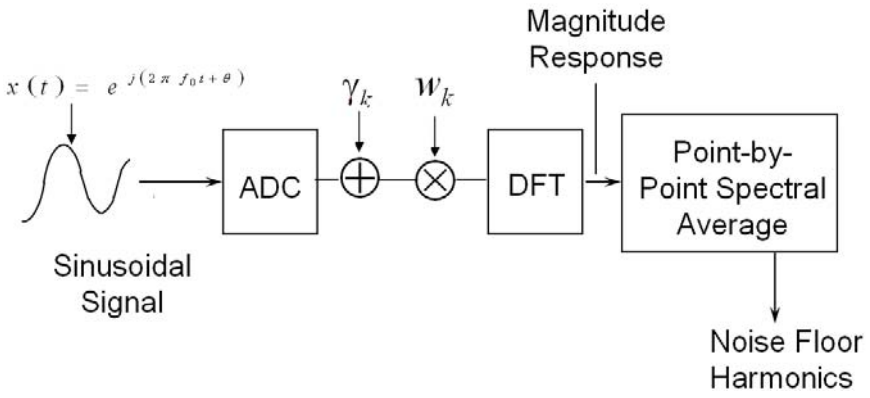

Figure 70. Process for Examining ADC Noise Floor (From [10]).

This process involves passing a sinusoid input signal through the ADC. During quantization, a white noise process (quantization noise) γ_k is added to the analog signal. The ADC noise floor is analyzed by windowing 20 sets of 4,096 digitized samples using a Blackman-Harris window, with the window samples represented by w_k. The Blackman-Harris window is chosen because of the low side-lobe levels it can achieve [10].

The average magnitude spectrum response is obtained by using the Discrete Fourier Transform (DFT) to transform the time-domain signals into the frequency domain and performing asynchronous point-by-point spectral averaging of the 20 sets of data.

The purpose of conducting this noise floor analysis is to determine the dominant noise sources in the ADC system, explained in the next two sections.

1. Quantization Noise

The theoretical noise floor can be evaluated by examining the SNR and considering the presence of quantization noise only. By normalizing the square of the

magnitude of the spectral average with the fundamental signal, the equation to calculate the noise floor using a Blackman-Harris window is [10]:

$$F_{M^2} = 10\log_{10}\left(\frac{3M}{4E_B}\right) + 6.02n \quad [\text{dB}] \tag{52}$$

where n = Number of ADC Bits = $log_2\ \hat{M}$ = log_2 (41) = 5.26 bits, E_B = Equivalent Noise Bandwidth of Blackman-Harris Window = 2.0, and M = Number of Samples = 4096. The theoretical noise floor is calculated as $F_{M2} = -64.13$ dB using (52). The procedure in Figure 70 is then used to obtain actual ADC noise floor measurements for comparison.

For this analysis, a 1-kHz sinusoidal signal was sampled at 100-kHz. Twenty sets of 4,096 digitized signals were acquired asynchronously and windowed using a Blackman-Harris window. The MATLAB Fast Fourier Transform (FFT) function was used to compute the signal spectrum. The point-by-point spectral average of the twenty sets of data was then calculated.

The spectral average of the 1-kHz sinusoidal signal is shown in Figure 71, using a Blackman-Harris window.

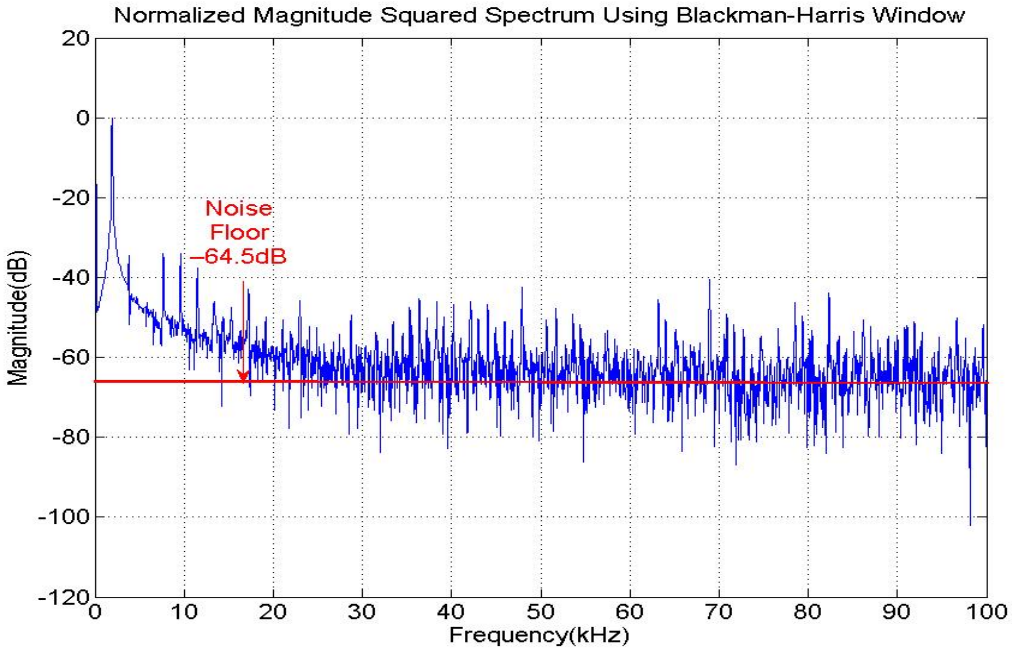

Figure 71. Spectral Average of a 1-kHz Sinusoidal Signal.

Since the noise floor in Figure 71 is –64.5 dB, which is close to the theoretical noise floor of –64.13 dB, other additive noise sources (such as thermal noise) are less dominant in the system compared to quantization noise.

2. Clock Jitter

Two frequencies can be used to test if clock jitter is a dominant noise source, with the higher frequency being twice that of the first frequency [10]. A 2-kHz sinusoidal signal was sampled to compare its noise floor with that obtained for the 1-kHz sinusoidal signal in Figure 71. The same process in Figure 70 was adhered to in calculating the magnitude square spectrum of both signals.

The spectral average of the 2-kHz sinusoidal signal with a noise floor of –59.5 dB, using a Blackman-Harris window, is shown in Figure 72. Since the difference between both noise floor levels is 5 dB (less than 6 dB), clock jitter is not the dominant noise source [10]. This is expected as clock jitter is not expected to be significant at these relatively low frequencies.

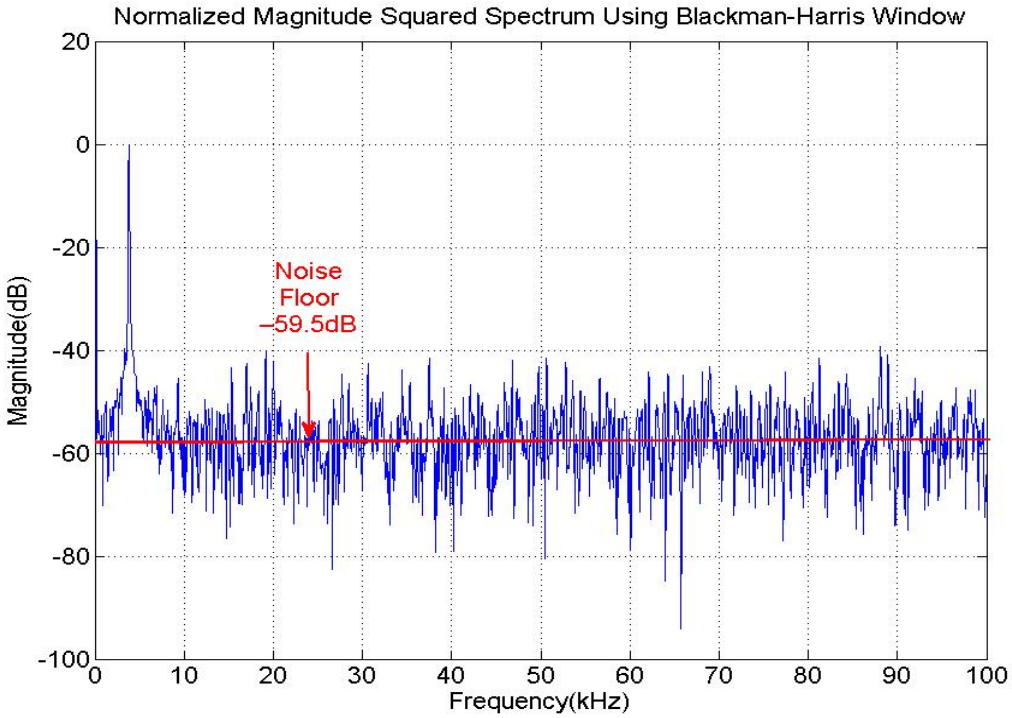

Figure 72. Spectral Average of a 2-kHz Sinusoidal Signal.

C. DYNAMIC PERFORMANCE ANALYSIS

Lastly, the ADC performance is characterized using the following dynamic performance parameters: SNR, THD, SINAD and ENOB. Two different frequencies (1-kHz and 2-kHz) were used for comparison.

1. Signal-to-Noise Ratio

The ideal SNR equation assumes only quantization noise and is expressed as [10]:

$$SNR(dB) = 6.02n + 1.76 \tag{53}$$

where $n =$ Number of ADC Bits $= \log_2 \hat{M} = \log_2(41) = 5.36$ bits.

2. Total Harmonic Distortion

The THD measures the harmonics of the input signal that show up at integral multiples of the fundamental frequency, and is a measure of the ADC's non-linearity. It is defined as [10]:

$$THD(-dB) = 20\log\sqrt{\left(10^{2^{nd}\ HAR/20}\right)^2 + \left(10^{3^{rd}\ HAR/20}\right)^2 \ldots} \tag{54}$$

For the purpose of this analysis, the input signal's first five harmonics were measured in decibels and used in (54).

3. SNR Plus Distortion

The SINAD equation takes into account all of the noise (including harmonics) to give an indication of the useful ADC dynamic range, but excludes the DC component. It is expressed as [10]:

$$SINAD(+dB) = -20\log\sqrt{10^{-(SNR)/10} + 10^{THD/10}}. \tag{55}$$

4. Effective Number of Bits

The ENOB is a measure of the usable ADC dynamic range, which is reduced due to noise. It is defined as [10]:

$$ENOB = \frac{SINAD - 1.76 + 20\log\left(\frac{\text{full scale amplitude}}{\text{actual input amplitude}}\right)}{6.02}. \quad (56)$$

5. Summary of ADC Dynamic Performance Parameters

A summary of the calculated parameter values, using (53) to (56) is shown in Table 7.

Table 7. ADC Dynamic Performance Parameters.

Frequency	Ideal SNR (dB)	SINAD (dB)	THD (dB)	ENOB (Bits)
1 kHz	34.22	34.21	–61.68	5.39
2 kHz	34.22	34.21	–61.01	5.39

From Table 7, it is observed that the THD for the 2-kHz signal is 0.67 dB higher than that for the 1-kHz signal. The SINAD and ENOB values are similar for both frequencies, indicating that there was insignificant signal distortion when the frequency is increased from 1-kHz to 2-kHz.

An enabler to achieving this is the design of comparator circuits in the FPGA, which has significantly reduced the distortion caused by sampling errors, as compared to using actual comparator ICs.

D. SUMMARY

In this chapter, the ADC performance was described in three ways. First, the ADC transfer function was examined for linearity errors. Second, the noise floor was analyzed to determine the main sources of noise in the system. Lastly, the dynamic performance parameters of the ADC system were characterized. Key conclusions and recommendations for future research will be provided in the next chapter.

THIS PAGE INTENTIONALLY LEFT BLANK

VIII. CONCLUSION

The key conclusions obtained from this project and recommendations for future research are provided in this chapter.

A. KEY CONCLUSIONS

The goal of this thesis was to conduct hardware and software implementation of the DDS module of the folding ADC architecture (for Moduli 7, 8, 9) from the bank of comparators to the RSNS-to-binary conversion within the FPGA as well as integration with the front-end PES module of this ADC design. This was accomplished via several milestones described below.

Firstly, the RSNS dynamic range computation algorithm in [6] and [7] was verified to be correct, proving that an eight-bit DR of 133 can be achieved theoretically for a three-channel RSNS ADC with Moduli $m_1 = 7$, $m_2 = 8$ and $m_3 = 9$.

Secondly, the RSNS-to-binary algorithm was implemented in LabVIEW and shown to achieve the DR value of 133, which is in agreement with [6] and [7], and a one-bit improvement over that achieved in [4]. Design of thermometer code generator circuits and simulation of this algorithm were carried out to verify that it is working properly before connecting to actual signals.

Thirdly, the comparator circuits and RSNS-to-binary conversion algorithm were implemented on the FPGA, allowing the ADC to achieve a higher sampling frequency.

Lastly, the DDS module was integrated with the front-end PES module in [5] to form a folding ADC system and characterization of the ADC performance was carried out. Analysis of the results attributed the dominant noise source in the ADC system to quantization noise, with the ADC remaining resilient to errors caused by other additive noise sources and comparator sampling.

This electro-optic RSNS ADC system has been demonstrated to work and produces a seven-bit output with relatively simple hardware and software. Due to the reduced number of hardware and software components and the large bandwidth of the photonics components, the energy and size savings, as well as increase in speed, make this folding ADC design appealing for defense applications, such as unmanned systems, direction-finding antenna architectures and electronic warfare system-on-a-chip applications.

B. RECOMMENDATIONS FOR FUTURE RESEARCH

While this thesis is concerned with optimizing the DDS module implementation of the folding ADC system, various system trade-offs had to be made for integration purposes. Further upgrades and research can be carried out in various aspects of the system, detailed below.

1. Bandwidth Upgrade

The bandwidth of this ADC system is currently limited by the NI-9215 Analog Input Module's data rate of 100 kS/s. The highest frequency it can sample is 50 kHz, based on the Nyquist criteria. There are several wider-bandwidth NI modules available to replace this, such as the NI-5761 Digitizer Adapter Module, which has a data rate of 250 MS/s and can sample frequencies up to 125 MHz [11]. The only drawback is that choices of wide-bandwidth modules are limited due to lack of commercial development and require impedance-matching for maximum power transfer.

2. FPGA Upgrade

The speed of this ADC system is currently limited by the XILINX Virtex-5 LX-30 FPGA on the NI-9111 Chassis. There are several higher-capacity NI FPGA modules available to replace this, such as the FlexRIO PXIe-7965R module, which has a XILINX Virtex-5 SX-95T FPGA with a clock speed of 550 MHz and the ability to handle single-ended Input/Output (I/O) up to 800 Mbps [12]. This module did not arrive in time for the project due to procurement delay, but can be easily substituted when it is available.

There are other high-speed FPGAs (up to 1 GHz) that could not be used as they do not interface with NI LabVIEW. Nevertheless, it is envisaged that an Application-Specific Integrated Circuit (ASIC) can be developed for testing once the FPGA circuit design has been fixed. This will remove the constraint of having to rely only on the NI programming environment.

The envisaged DDS setup, after incorporating the component upgrades, is illustrated in Figure 73.

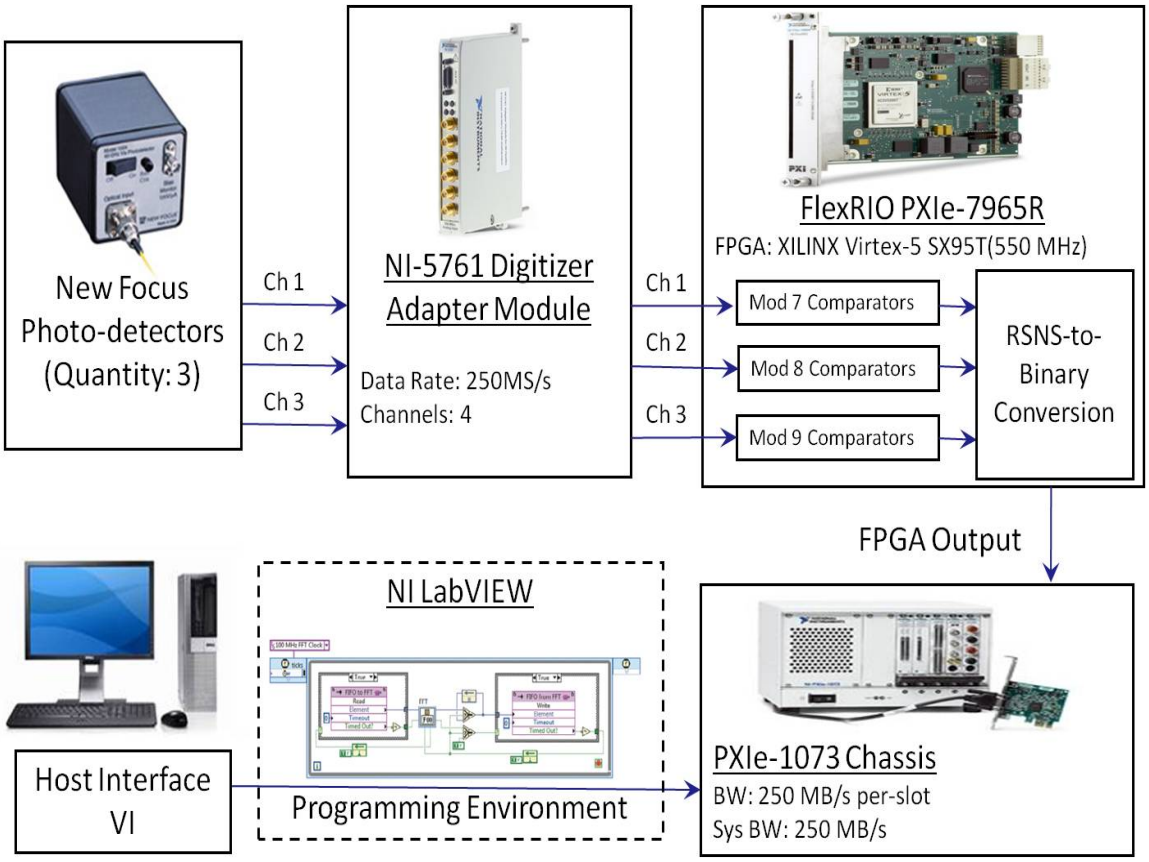

Figure 73. Upgrades to DDS Setup.

3. Higher-Moduli Configurations

The modular structure of the RSNS-to-binary converter allows the ADC system to be easily scalable to higher-moduli configurations or configurations with more than three channels. This will increase the ADC resolution and enable the ADC system to achieve a dynamic range that is better than the current eight-bit DR of 133.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX. GENERALIZED CHINESE REMAINDER THEOREM PROCEDURE TO SOLVE FOR COA SHIFTS

The COA shift h_{s1} is the least positive solution to the following sets of congruence equations:

$$\begin{aligned}
&\frac{h_{s1}-1}{3} \equiv 0 \,(\text{mod } 7) \longrightarrow h_{s1} \equiv 1 \,(\text{mod } 21) \\
&\frac{h_{s1}-1}{3} \equiv 0 \,(\text{mod } 8) \longrightarrow h_{s1} \equiv 1 \,(\text{mod } 24) \\
&\frac{h_{s1}+2}{3} \equiv 0 \,(\text{mod } 9) \longrightarrow h_{s1} \equiv -2 \,(\text{mod } 27)
\end{aligned} \tag{57}$$

Equation (57) cannot be solved directly using the standardized Chinese Remainder Theorem (CRT) as the three moduli are not Pair-Wise Relatively Prime (PRP). The generalized CRT procedure must be used. The first step is to break each equation into its constituent equations [10]:

$$h_{s1} \equiv \; 1 \,(\text{mod } 21) \begin{cases} h_{s1} \equiv 1 \,(\text{mod } 7) \\ h_{s1} \equiv 1 \,(\text{mod } 3) \end{cases}$$

$$h_{s1} \equiv \; 1 \,(\text{mod } 24) \begin{cases} h_{s1} \equiv 1 \,(\text{mod } 8) \\ h_{s1} \equiv 1 \,(\text{mod } 3) \end{cases} \tag{58}$$

$$h_{s1} \equiv -2 \,(\text{mod } 27) \begin{cases} h_{s1} \equiv -2 \,(\text{mod } 3) \\ h_{s1} \equiv -2 \,(\text{mod } 3) \\ h_{s1} \equiv -2 \,(\text{mod } 3) \end{cases}$$

The second step is to group all constituent equations with the same moduli together, and solve for the remaining congruence equations using the standardized CRT method:

$$\begin{aligned}
&h_{s1} \equiv 1 \,(\text{mod } 7) \\
&h_{s1} \equiv 1 \,(\text{mod } 8) \\
&h_{s1} \equiv -2 \,(\text{mod } 3)
\end{aligned} \tag{59}$$

The standardized CRT solution is:

$$h_{s1} = \sum_{i=1}^{N} \frac{M}{m_i} b_i a_i$$

$$\text{where } M = \prod m_i = (7)(8)(3) = 168 \tag{60}$$

$$m_i = \text{ Moduli } i = [m_1 \quad m_2 \quad m_3] = [7 \quad 8 \quad 3]$$

$$a_i = \text{ Residue of Moduli } i = [a_1 \quad a_2 \quad a_3] = [1 \quad 1 \quad -2]$$

The coefficients b_i can be found via the following procedure:

Find $b_1 : \left(\frac{M}{m_1}, m_1\right) = \left(\frac{168}{7}, 7\right) = (24, 7)$

$$\left.\begin{aligned} 24(1) + 7(0) &= 24 \\ 24(0) + 7(1) &= 7 \end{aligned}\right\} \left\lfloor \frac{24}{7} \right\rfloor = 3$$

$$\left.\begin{aligned} 24\left[(1) - (0)\left(\left\lfloor \frac{24}{7} \right\rfloor\right)\right] + 7\left[(0) - (1)\left(\left\lfloor \frac{24}{7} \right\rfloor\right)\right] &= 24(1) + 7(-3) = 3 \\ 24\left[(0) - (1)\left(\left\lfloor \frac{7}{3} \right\rfloor\right)\right] + 7\left[(1) - (-3)\left(\left\lfloor \frac{7}{3} \right\rfloor\right)\right] &= 24(-2) + 7(7) = 1 \end{aligned}\right\} \left\lfloor \frac{7}{3} \right\rfloor = 2 \quad \therefore b_1 = -2$$

Find $b_2 : \left(\frac{M}{m_2}, m_2\right) = \left(\frac{168}{8}, 8\right) = (21, 8)$

$$\left.\begin{aligned} 21(1) + 8(0) &= 21 \\ 21(0) + 8(1) &= 8 \end{aligned}\right\} \left\lfloor \frac{21}{8} \right\rfloor = 2$$

$$\left.\begin{aligned} 21\left[(1) - (0)\left(\left\lfloor \frac{21}{8} \right\rfloor\right)\right] + 8\left[(0) - (1)\left(\left\lfloor \frac{21}{8} \right\rfloor\right)\right] &= 21(1) + 8(-2) = 5 \\ 21\left[(0) - (1)\left(\left\lfloor \frac{8}{5} \right\rfloor\right)\right] + 8\left[(1) - (-2)\left(\left\lfloor \frac{8}{5} \right\rfloor\right)\right] &= 21(-1) + 8(3) = 3 \end{aligned}\right\} \left\lfloor \frac{8}{5} \right\rfloor = 1$$

$$\left.\begin{aligned} 21\left[(1) - (-1)\left(\left\lfloor \frac{5}{3} \right\rfloor\right)\right] + 8\left[(-2) - (3)\left(\left\lfloor \frac{5}{3} \right\rfloor\right)\right] &= 21(2) + 8(-5) = 2 \\ 21\left[(-1) - (2)\left(\left\lfloor \frac{3}{2} \right\rfloor\right)\right] + 8\left[(3) - (-5)\left(\left\lfloor \frac{3}{2} \right\rfloor\right)\right] &= 21(-3) + 8(8) = 1 \end{aligned}\right\} \left\lfloor \frac{3}{2} \right\rfloor = 1 \quad \therefore b_2 = -3$$

$$\text{Find } b_3 : \left(\frac{M}{m_3}, m_3\right) = \left(\frac{168}{3}, 3\right) = (56, 3)$$

$$\left.\begin{aligned} 56(1) + 3(0) &= 56 \\ 56(0) + 3(1) &= 3 \end{aligned}\right\} \left\lfloor \frac{56}{3} \right\rfloor = 18$$

$$\left.\begin{aligned} 56\left[(1) - (0)\left(\left\lfloor \frac{56}{3} \right\rfloor\right)\right] + 3\left[(0) - (1)\left(\left\lfloor \frac{56}{3} \right\rfloor\right)\right] &= 56(1) + 3(-18) = 2 \\ 56\left[(0) - (1)\left(\left\lfloor \frac{3}{2} \right\rfloor\right)\right] + 3\left[(1) - (-18)\left(\left\lfloor \frac{3}{2} \right\rfloor\right)\right] &= 56(-1) + 3(19) = 1 \end{aligned}\right\} \left\lfloor \frac{3}{2} \right\rfloor = 1 \quad \therefore b_3 = -1$$

$$h_{s1} = \left(\frac{M}{m_1} b_1 a_1 + \frac{M}{m_2} b_2 a_2 + \frac{M}{m_3} b_3 a_3\right) (\text{mod } M)$$

$$\therefore h_{s1} = \left[24(-2)(1) + 21(-3)(1) + 56(-1)(-2)\right] (\text{mod } 168) = 1 (\text{mod } 168)$$

Similarly, the COA shift h_{s2} is the least positive solution to the following sets of congruence equations:

$$\begin{aligned} \frac{h_{s2} - 2}{3} &\equiv 0 \text{ (mod } 7) \longrightarrow h_{s2} \equiv 2 \text{ (mod } 21) \\ \frac{h_{s2} + 1}{3} &\equiv 0 \text{ (mod } 8) \longrightarrow h_{s2} \equiv -1 \text{ (mod } 24) \\ \frac{h_{s2} + 1}{3} &\equiv 0 \text{ (mod } 9) \longrightarrow h_{s2} \equiv -1 \text{ (mod } 27) \end{aligned} \tag{61}$$

Equation (61) cannot be solved directly using the standardized CRT as the three moduli are not PRP. The generalized CRT procedure must be used. Again, the first step is to break each equation into its constituent equations [10]:

$$h_{s2} \equiv \ \ 2 \,(\text{mod}\ \ 21) \begin{cases} h_{s2} \equiv 2 \,(\text{mod}\ \ 7) \\ h_{s2} \equiv 2 \,(\text{mod}\ \ 3) \end{cases}$$

$$h_{s2} \equiv -1 \,(\text{mod}\ \ 24) \begin{cases} h_{s2} \equiv -1 \,(\text{mod}\ \ 8) \\ h_{s2} \equiv -1 \,(\text{mod}\ \ 3) \end{cases} \tag{62}$$

$$h_{s2} \equiv -1 \,(\text{mod}\ \ 27) \begin{cases} h_{s2} \equiv -1 \,(\text{mod}\ \ 3) \\ h_{s2} \equiv -1 \,(\text{mod}\ \ 3) \\ h_{s2} \equiv -1 \,(\text{mod}\ \ 3) \end{cases}$$

The second step is to group all constituent equations with the same moduli together and solve for the remaining congruence equations using the standardized CRT method:

$$\begin{aligned} h_{s2} &\equiv 2 \,(\text{mod}\ \ 7) \\ h_{s2} &\equiv -1 \,(\text{mod}\ \ 8) \\ h_{s2} &\equiv -1 \,(\text{mod}\ \ 3) \end{aligned} \tag{63}$$

The standardized CRT solution is:

$$\begin{aligned} &h_{s2} = \sum_{i=1}^{N} \frac{M}{m_i} b_i a_i \\ &\text{where } M = \prod m_i = (7)(8)(3) = 168 \\ &\qquad m_i = \text{ Moduli } i = \begin{bmatrix} m_1 & m_2 & m_3 \end{bmatrix} = \begin{bmatrix} 7 & 8 & 3 \end{bmatrix} \\ &\qquad a_i = \text{ Residue of Moduli } i = \begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix} = \begin{bmatrix} 2 & -1 & -1 \end{bmatrix} \end{aligned} \tag{64}$$

The coefficients b_i are the same as for h_{s1}, as the moduli configuration is the same for both COA shifts:

$$\begin{aligned} &h_{s2} = \left(\frac{M}{m_1} b_1 a_1 + \frac{M}{m_2} b_2 a_2 + \frac{M}{m_3} b_3 a_3 \right) (\text{mod}\, M) \\ &\therefore h_{s2} = \left[24(-2)(2) + 21(-3)(-1) + 56(-1)(-1) \right] (\text{mod}\ 168) = 23 (\text{mod}\ 168) \end{aligned} \tag{65}$$

LIST OF REFERENCES

[1] P. E. Pace, D. Styer and I. A. Akin, "A folding ADC preprocessing architecture employing a robust symmetrical number system with gray-code properties," *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, vol. 47, pp. 462–467, May 2000.

[2] B. L. Luke, "Architecture of an integrated microelectronic warfare system on a chip and design of key components," PhD dissertation, Naval Postgraduate School, Monterey, California, December 2004.

[3] M. R. Arvizo, "Electro-optic analog-to-digital converter (ADC) architecture based on the robust symmetrical number system (RSNS)," M.S. thesis, Naval Postgraduate School, Monterey, California, September 2009.

[4] R. A. Monta, "Symmetrical residue-to-binary conversion algorithm, pipelined FPGA implementation, and testing logic for use in high-speed folding digitizers," M.S. thesis, Naval Postgraduate School, Monterey, California, December 2005.

[5] K. L. Tong, "Photonic analog-to-digital converters preprocessing using the robust symmetrical number system antenna signals," M.S. thesis, Naval Postgraduate School, Monterey, California, December 2010.

[6] B. L. Luke and P. E. Pace, "*N*-Sequence RSNS ambiguity analysis," *IEEE Transactions on Information Theory*, vol. 53, pp. 1759–1766, May 2007.

[7] B. L. Luke and P. E. Pace, "Computation of the Robust Symmetrical Number System Dynamic Range," IEEE Information Theory Workshop (ITW 2010), Dublin, Ireland, September 2010.

[8] D. Styer and P. E. Pace, "Two-channel RSNS dynamic range," *IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications*, vol. 49, pp. 395–397, March 2002.

[9] B. L. Luke and P. E. Pace, "RSNS-to-binary conversion," *IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications*, vol. 54, pp. 2030–2043, September 2007.

[10] Pace, P. E., *Advanced Techniques for Digital Receivers*. Norwood, MA: Artech House, 2000.

[11] National Instruments: "14-bit, 250MS/s Adapter Module for NI FlexRIO Data Sheet," July 2010. [Available online at http://sine.ni.com/ds/app/doc/p/id/ds-236/lang/en, accessed 27 Oct 2010].

[12] National Instruments: “NI FlexRIO FPGA Modules,” July 2010. [Available online at http://www.ni.com/pdf/products/us/cat_flexriofpga.pdf, accessed 27 Oct 2010].

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Fort Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Chairman, Code ECE
   Naval Postgraduate School
   Monterey, California

4. Professor Phillip Pace
   Naval Postgraduate School
   Monterey, California

5. Professor David Jenn
   Naval Postgraduate School
   Monterey, California

6. Dr. James Calusdian
   Naval Postgraduate School
   Monterey, California

7. Professor Tat Soon Yeo
   Temasek Defence Systems Institute
   Singapore

8. Miss Lai Poh Tan
   Temasek Defence Systems Institute
   Singapore